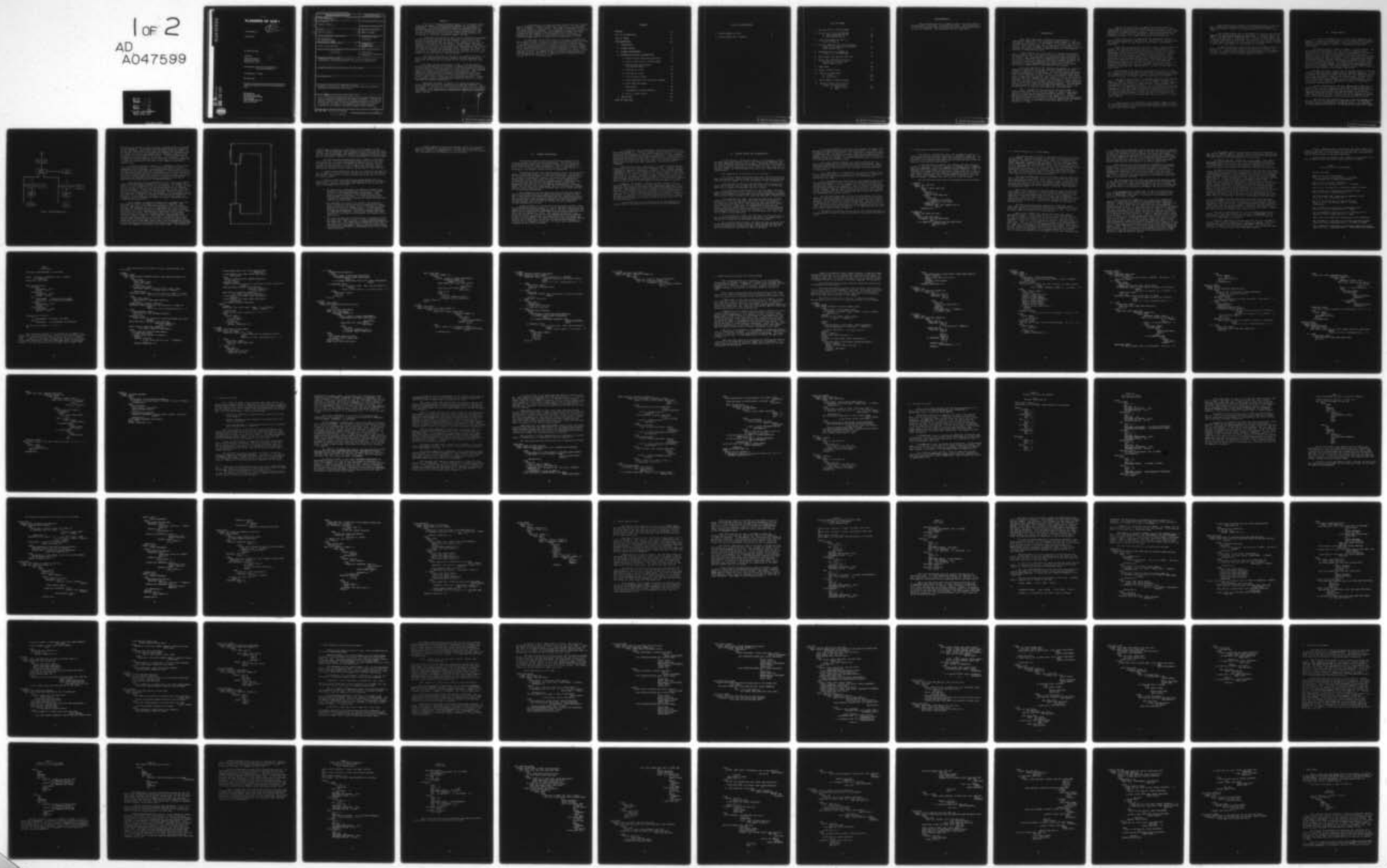
AD-A047 599 SRI INTERNATIONAL MENLO PARK CALIF F/G 5/2
PLANNERS OF ACS.1.(U)
NOV 77 M C PEASE N00014-77-C-0308
UNCLASSIFIED SRI-TR-15 NL

1 OF 2
AD
A047599



PLANNERS OF ACS.1

12

Technical Report 15

November 1977

By: Marshall C. Pease

Prepared for:

Office of Naval Research
Department of the Navy
Arlington, Virginia 22217

Contract Monitor: Marvin Denicoff, Program Director
Information Systems Branch

Contract N00014-77-C-0308

SRI Project 6289

332 500

SRI International
333 Ravenswood Avenue
Menlo Park, California 94025
(415) 326-6200
Cable: STANRES, Menlo Park
TWX: 910-373-1246

SRI International

SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

REPORT DOCUMENTATION PAGE		READ INSTRUCTIONS BEFORE COMPLETING FORM
1. REPORT NUMBER (9) Technical Report No. 15	2. GOVT ACCESSION NO.	3. RECIPIENT'S CATALOG NUMBER
4. TITLE (and Subtitle) (6) Planners of ACS.1 .		5. TYPE OF REPORT & PERIOD COVERED
		6. PERFORMING ORG. REPORT NUMBER
7. AUTHOR(s) (10) Marshall C. Pease		8. CONTRACT OR GRANT NUMBER(s) (15) N00014-77-C-0308
9. PERFORMING ORGANIZATION NAME AND ADDRESS SRI International 333 Ravenswood Avenue Menlo Park, CA 94025		10. PROGRAM ELEMENT, PROJECT, TASK AREA & WORK UNIT NUMBERS NR 049-308
11. CONTROLLING OFFICE NAME AND ADDRESS Office of Naval Research		12. REPORT DATE (11) November 1977
		13. NUMBER OF PAGES 123
14. MONITORING AGENCY NAME & ADDRESS(if different from Controlling Office) (12) 134 p.		15. SECURITY CLASS. (of this report) Unclassified
		15a. DECLASSIFICATION/DOWNGRADING SCHEDULE

16. DISTRIBUTION STATEMENT (of this Report)
Distribution of this document is unlimited. It may be released to the Clearinghouse, Department of Commerce, for sale to the general public.

(14) SRI-TR-15

17. DISTRIBUTION STATEMENT (of the abstract entered in Block 20, if different from Report)

18. SUPPLEMENTARY NOTES

19. KEY WORDS (Continue on reverse side if necessary and identify by block number)
Management, knowledge-base, modularity, planning, replanning coordination, organization, model process model, demons.

20. ABSTRACT (Continue on reverse side if necessary and identify by block number)
ACS.1, for "Automated Command Support," is a research system studying the design of knowledge-based systems for the support of management. The research addresses managerial responsibility for planning operations, maintaining and executing approved plans, and for the retrospective analysis of the results of operations. The purpose of the research is to develop architectural principles for the design of intelligent management support systems.

DD FORM 1 JAN 73 1473 EDITION OF 1 NOV 65 IS OBSOLETE

410 281

SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

ABSTRACT

ACS.1, for "Automated Command Support," is a research system studying the design of knowledge-based systems for the support of management. The research addresses managerial responsibility for planning operations, maintaining and executing approved plans, and for the retrospective analysis of the results of operations. The purpose of the research is to develop architectural principles for the design of intelligent management support systems.

Viewed from the top level, ACS.1 can be regarded as an assembly of modules called "schedulers" and "planners," with certain other modules and subsystems that are not of direct concern here. The planners "know how to" plan operations. The schedulers "know how to" coordinate the expected use of resources, whether human, equipment, supplies, or facilities. The planners have responsibility for creating detailed plans to meet specified objectives, including the timing of all required tasks and the assignment of all necessary resources. The schedulers are responsible for assigning the resources so as to avoid conflict with other plans or expected events.

This report describes the design of the planners of ACS.1. It details the functions that create a planner and that implement its operations and discusses the reasons for the choices that have been made.

The major problem of planning in ACS.1 can be described as providing mechanisms that will enforce a complex and interlocking set of constraints as conditions change or as new requirements are entered. A plan is viewed, in this context, as a data structure. The knowledge used by the planner provides one set of constraints on a plan. Additional constraints are provided by the schedulers and by other planners that may be called on to plan tasks within a given plan. Current time is also a constraint on the planning operation. A plan, to be workable, must be simultaneously consistent with all of these expressed and implied constraints.

Management problems, in general, are characterized by the extent and diversity of the constraints that must be satisfied. The development of techniques for maintaining consistency within a changing environment, where consistency is defined by a complex set of exogenic constraints, is an important step towards providing useful managerial support systems.

In the planners, the constraints are enforced through a complex system of demons. A demon, as the term is used here, is a structure that is attached to a data element. The demon is examined whenever that data element is changed. If its preconditions are met, a function is called that may modify other data elements as needed to maintain consistency with respect to the constraint that it enforces. The planners of ACS.1 demonstrate that demons can be used to enforce continuing compliance to a complex and interlocking set of rules.

An additional objective for ACS.1 is to make it possible for the manager to adapt the system to changing requirements and situations. It is assumed that his needs will change, and that no system that cannot be easily modified will continue to provide effective support. One of the design principles that has been explored in detail in ACS.1 is the explicit encoding of the system's knowledge as a set of models that are available for adaptation by the manager or his staff. In the planners of ACS.1, the models are what are called "process models." The implementation used in ACS.1 demonstrates the feasibility of the concept and of the techniques used.

CONTENTS

LIST OF ILLUSTRATIONS

LIST OF TABLES

ACKNOWLEDGMENTS

The work described here is based, in part, on earlier work by Richard Fikes, Steven Weyl, and Kazuka Tachibana. Daniel Sagalowicz is currently participating in the program, and has contributed much to its current thinking. Jack Goldberg is project supervisor.

I. INTRODUCTION

This report describes the modules called planners in the experimental system ACS.1 (for Automated Command Support). ACS.1 is intended as a vehicle for the development of techniques for building knowledge-based systems that will provide intelligent support to a manager. The areas of support being addressed are planning operations, administration and monitoring of approved plans, and retrospective analysis of completed operations.

Viewed from the top level, ACS.1 is a system of autonomous modules, some of which are called "planners" and some "schedulers." The planners have the responsibility of planning specified types of activities. The schedulers are responsible for coordinating the use of particular types of resources, which may be people, equipment, or facilities. This report addresses the design of the planners, describing the other components of the system only to the extent necessary to understand the requirements on the planners.

ACS.1 has operated in the simulated environment of a naval air squadron, although the techniques used lend themselves to a wide variety of application environments. The principle operations being planned and managed are flight missions. This requires coordinating such various resources as pilots, aircraft, maintenance crews, deck crews, launch facility and crew, and recovery facilities and personnel. There are additional demands on these resources, such as the need of personnel for rest, or of equipment for maintenance. Other events can limit the availability of certain resources, such as a pilot becoming sick or an aircraft requiring unexpected maintenance. The development and maintenance of a plan to meet specified conditions require the satisfaction of an extensive and complex set of constraints.

ACS.1 is capable of dealing with an arbitrary number of resource types and with an arbitrary number of types of operations using the resources, limited only by machine capacity. Different application environments (squadron, wing, Tycom, etc.) require addressing different levels of detail and different policy constraints. The specialization of the system to different application environments and level of detail is obtained by specifying models that are appropriate to them.

Some of the constraints are imposed by the fact that the various types of resources are limited, and that there may be other commitments for these resources. The coordination needed to meet these constraints is handled by the schedulers. The detailed design of the schedulers of ACS.1 has been described in Technical Report 14, "The Schedulers of ACS.1" (Oct. 1977) and will not be described here.

Other constraints are imposed by the complexities of the operation being planned or administered. These are the responsibility of a planner, and it is with them that this report is principally concerned.

Note that the constraints interlock. While a planner seeks to satisfy those constraints that are due to the complexities of the operation, it must do so in a way that does not conflict with other commitments that have been made for the required resources. Further, the planner may not be able to recognize that a potential conflict exists until it has completed much of the planning process. The resolution of the resultant conflict by a scheduler may invalidate part of the plan that has been developed. The planner must then be prepared to revise what it has done to meet this new condition. Developing a complete plan involves not only the ability to meet its internal constraints, but also to respond to unexpected external conditions that may force replanning of all or part of the plan as it then exists.

The problem in the design of the planners of ACS.1 is, then, the construction of mechanisms that will enforce consistency according to a complex set of constraints, and that will continue to enforce it as new constraints are imposed from the schedulers or elsewhere.

The device used in ACS.1 for imposing and maintaining consistency is a complex system of what are called "demons." A demon is a structure that is attached to particular data items, and that contains a precondition and a function. When the data item is changed, the precondition is automatically checked. If it is found to be true, the function is called with specified variables and does what it has been programmed to do. The demon is then said to have been "fired." *
A major topic addressed in this report is the means used to construct the complex system of demons, and the way they act to maintain the consistency of a plan as it is generated, and as new data are entered into a planner.

----

* To be precise, this describes a write demon. There can also be read demons that may be fired on reading the data element. We have not used read demons, and will use the term "demon" without qualification to mean a write demon.

The problem can be regarded as developing procedures that will map a complex set of passive rules into a corresponding system of operators that will enforce continuing compliance of the data to the rules.

The system concept, and its development in terms of the desired application, has been described in some detail in Technical Report 13, "ACS.1: An Experimental Management Tool" (1977). In that report, the relation of this work to other research in artificial intelligence and in optimal scheduling is discussed. That material will not be repeated in this report, although a brief overview of the system concept is given to place the requirements for the schedulers in context.

In the next section, a brief overview of the system concept is given in order to place the design of the planners in context. In Section III, a general discussion of the requirements for the planners of ACS.1 is given in order to define more precisely what is needed. Finally, in Section IV, the detailed design is given.

II. SYSTEM CONCEPT

ACS.1 has been implemented on a PDP-10 using INTERLISP under TOPS-20. It uses a simulated clock in an interactive mode in order to simulate an operational environment. While its speed of response is dependent on the computer loading, it is generally limited by input/output.

The system concept is shown in Figure 1 in block diagram. The main elements of the system are a set of virtual modules called "planners" and "schedulers," each of which is responsible for a well defined part of the system's operation--a planner for planning a specific type of activity and a scheduler for coordinating the planned usage of a specific type of resource. Interactions among the modules are handled entirely through messages that are passed through a unit called the "message handler." All communications to or from a user, or to or from the data system, also pass through the message handler. The use of the message handler is an important feature helping to maintain the autonomy of the separate modules and providing a central switch for user control of the system's operations.

Between the user's terminal and the message handler is the user interface, which provides a pseudo-natural language capability. It uses a language facility called LIFER, developed for other purposes by the Artificial Intelligence Center of SRI International. The user interface accepts requests or commands in natural language format. It is not a true natural language capability; the unit uses pattern recognition rather than syntactic and semantic analysis. It provides a convenient interface facility.

The data system has not yet been implemented in more than a marginal way. It is intended to be more than a simple repository of data; it will have the responsibility for monitoring the execution of approved plans, checking that the required tasks are started and completed according to plan, and that the needed resources are available. When conflict occurs, it will initiate the appropriate system action, such as replanning or advising the user of the situation.

The planners and schedulers each use a body of knowledge that defines how it is to exercise its responsibilities. The knowledge used by a planner describes the process for which it is responsible to a specified level of detail. It identifies the tasks that compose

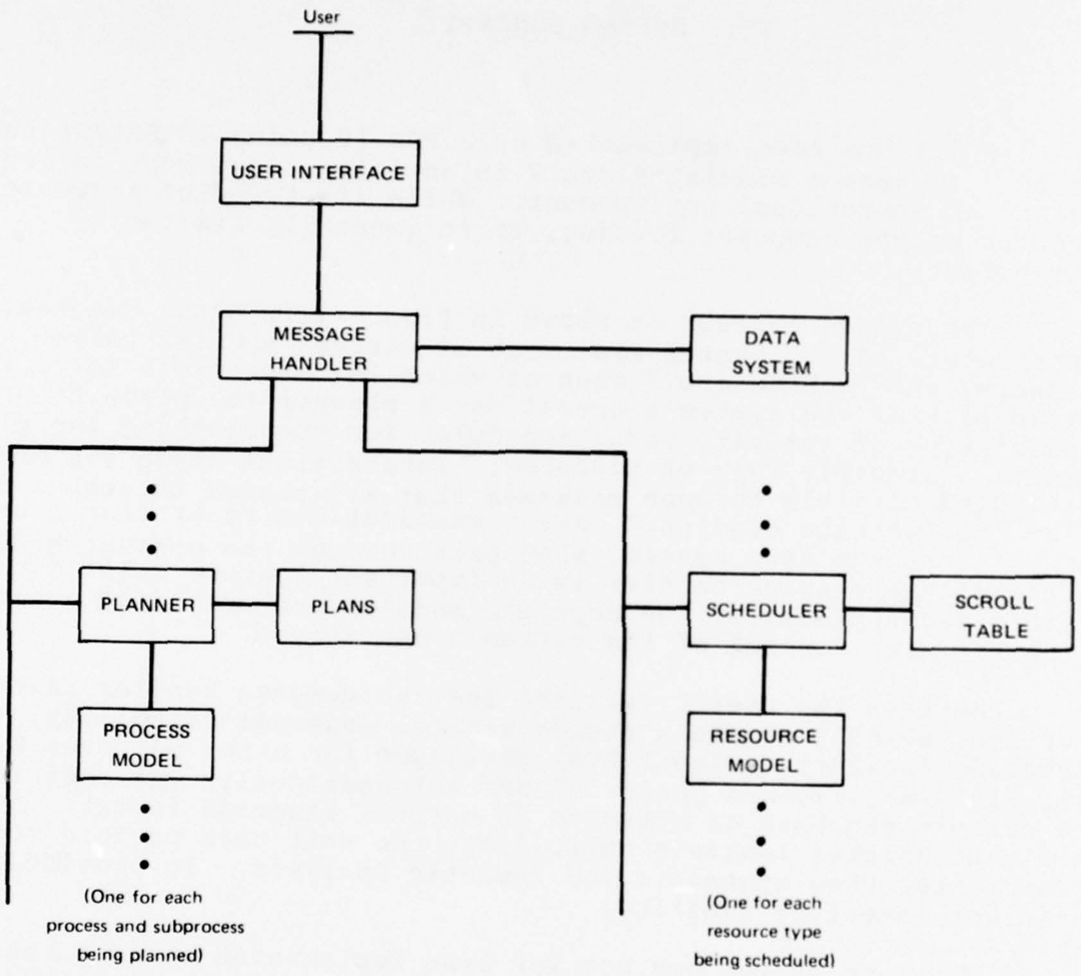


FIGURE 1 BLOCK DIAGRAM OF ACS.I

the process at the given level of detail, and what kinds of resources must be assigned. It describes the constraints among the tasks such as the requirement that certain tasks be completed before others can be started. It describes the constraints between the assignments and the tasks, such as the requirement that an assignment span the times during which certain tasks are being done. It also includes how an assignment can be obtained, or a plan for a task requested. That is, it knows how to make requests of the schedulers and other planners and it knows the identities of the ones it must consult to generate a complete plan.

A task recognized by a planner as a component in its process may itself require planning. If so, some other planner has the responsibility for planning it, perhaps decomposing it further into subtasks and obtaining additional assignments. The system of planners, in responding to some particular requirement, may be structured into a hierarchy of modules operating at various levels of detail. Different hierarchies may be required in response to different requirements, as implied by the knowledge contained in the planners and as may be modified by directions given to the message handler.

Each scheduler has the responsibility for a particular type of resource, whether human, equipment, or facilities. In response to a request for the assignment of one of the resources of its type for some future interval of time, it first *determines* which of its resources will be available. If more than one is available, and if it has been given the authority, it makes a selection and returns the name. It also enters the commitment into its own data. If it has not been given the authority, it sends the relevant information to the manager for his decision. If no resource is available, it will either return the closest available assignment or refuse the request, depending on the responsibility given to it.

As an example, consider the application environment that has been studied--that of the command of a naval air squadron. The principle type of activity that needs planning is that of flying a mission. The commander may enter a requirement that a given mission be planned to leave the ship at some specified time and to return at some later time. The requirement is accepted by a planner that "knows how to" plan a mission. The knowledge contained in that planner, encoded in what is called its "process model," may decompose the activity as shown in Figure 2. The tasks it recognizes are those of the preflight preparation of the aircraft, briefing the pilot, the flight itself, the postflight service of the aircraft, and the pilot debriefing. (The actual model used in ACS.1 also decomposed the flight into the flight out, the action at the target area, and the flight back, so that the commander could specify the strike times. The simplified

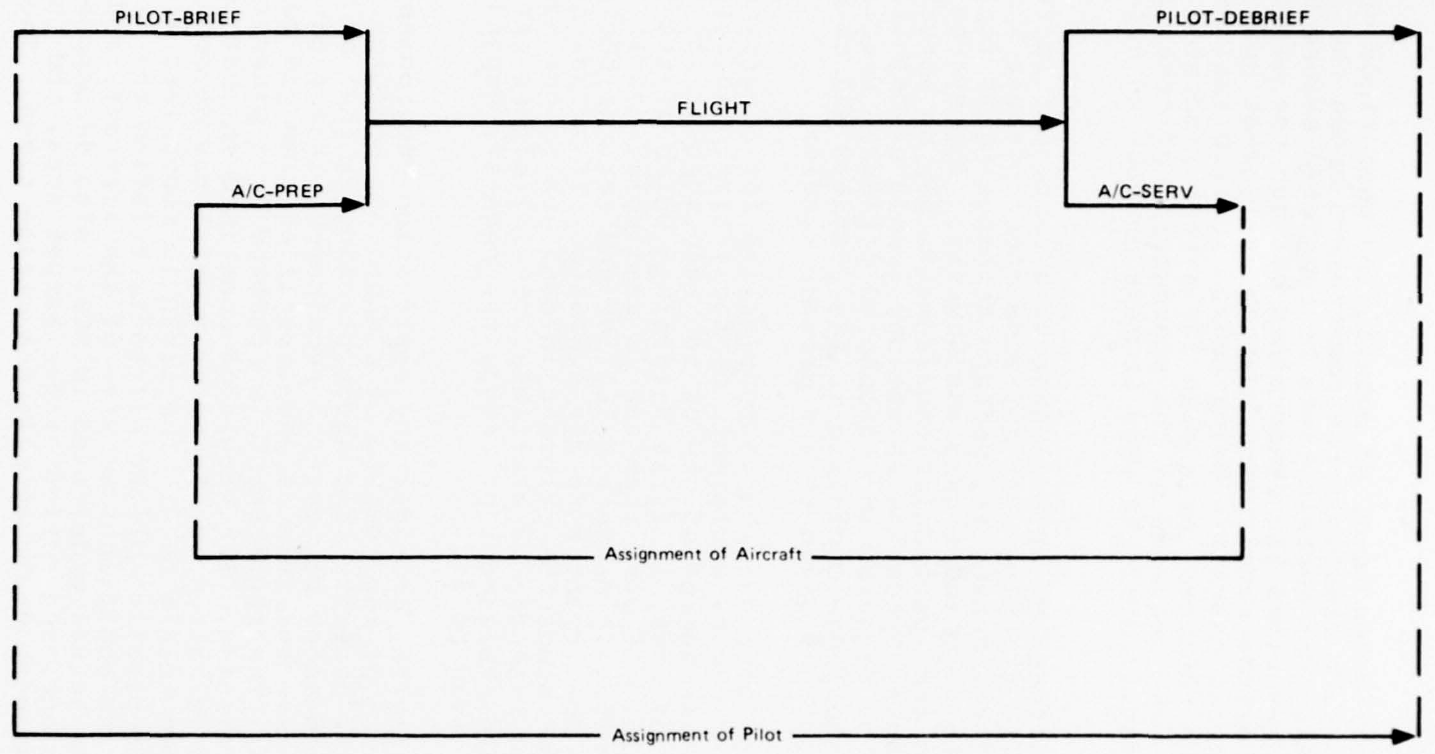


FIGURE 2 PROCESS MODEL FOR A MISSION

version shown in Figure 2 is sufficient for the purposes of this report.) The process model also includes the fact that a pilot and an aircraft must be assigned. The sequential constraints among the tasks, and the concurrency constraints between the assignments and the tasks that are indicated in Figure 2, are also contained in the process model.

The process model suggested by Figure 2 implies a particular level of detail. The task of preflight preparation of the aircraft, for example, may be further decomposed by another planner into the transfer of the aircraft to the flight deck, its preflight service, arming, and fueling, and its launch. Additional resources, such as maintenance personnel, may be required during some of these subtasks.

A plan has been generated, and will be returned to the commander for his approval or modification, only when all tasks and subtasks have been planned and when all resources needed during any task or subtask have been assigned.

There are several features of the system concept that are considered to be vital. These have dominated the design of the experimental system, and have been significant components of the research. They can be summarized as follows:

* The division of responsibilities among the planners and schedulers should correspond to the division of responsibilities in the comparable human organization. This permits an orderly growth of the system and facilitates understanding its operations. It also facilitates the transfer of responsibility between the system and the human organization when needed.

* The knowledge contained by the planners and schedulers should be explicit and accessible for modification without major revision of the system. This is necessary to permit adapting the system to changing needs. It also permits introducing new planners or schedulers through specifying the applicable knowledge. This feature permits the rapid extension of the scope of the system or its transfer to new application.

* The scope and operation of each planner or scheduler should be sufficiently simple to make it readily understandable by the human user. The complexity of system operation should be the result of interactions among the modules, rather than contained within any module. Again, this facilitates growth and adaptation, and permits rapid modification to meet exceptional conditions.

Further details of the system concept, and of the techniques that have been used to implement it, have been given in Technical Report 13, and are not repeated here. In the next section, we consider the requirements for the planners in greater detail.

III. PLANNER REQUIREMENTS

The principal system function served by the planners is the creation of plans to meet specified objectives. The objectives may be specified by a human user, by another planner, or in response to a need recognized by the system according to criteria that have been given to it. A closely related function is that of replanning-- i.e., revising an existing plan to meet altered circumstances.

The possibility that the objective may be set by another planner permits a plan's development to proceed hierarchically. For example, in the process model of Figure 2, the task of preflight preparation of the aircraft is recognized. However, the planner that contains this process model cannot itself plan this task. It can establish the requirement that the task shall be completed before a specified time, but its starting time will be developed by another planner that decomposes this task into a set of subtasks according to its process model. The action of the second planner will be initiated by a request from the first, giving the required launch time as the latest end time for the preflight preparation.

The possibility that the objective may be set by other system functions is illustrated in the application context of ACS.1 by the way it could enforce scheduled maintenance policy. The policy directs that certain maintenance actions be done when accumulated flight hours exceed some value. The accumulated flight hours can be maintained in the data system. When the limit is reached, the data system can initiate planning for the job.

Another example of a system-initiated planning activity is that of replanning when events require it. One of the major functions of the system is that of maintaining approved plans. By this is meant that the system shall recognize when events make an existing plan unworkable and shall take the appropriate action. The required action may be to replan the action, or to issue an alert message advising the commander of the situation, depending on the authority given to the system and encoded in its knowledge. The events that make a plan unworkable may be external to the plan, such as that an aircraft needs maintenance, or internal due to slippage in execution. In either case, replanning is triggered by the entry of new data.

It is important that replanning be confined to the part of a plan that requires it. Failure to observe this principle will not only lead to unnecessary work, it may actually upset the process. Complete replanning will temporarily release all assignments. Other planning activities can then intervene, taking control of the released resources and preventing recovery of the plan. While there are other ways to avoid this difficulty, the most direct way is to confine replanning to those parts of a plan that need replanning, leaving other parts untouched.

The ability to limit replanning is a result of basing system organization on a clean division of responsibility. A module, whether a planner or scheduler, responds to a request. This response becomes a commitment, and it is the responsibility of that module to meet the commitment--or to inform the source of the request that it is no longer able to do so. If data received by a module indicates the need for replanning by that module, it will attempt to revise the plan in such a way as to meet its original commitment. Only if it cannot do so will it pass the word up that a larger segment of the plan must be revised.

Suppose, for example, a mission has been planned. If the preflight service of the aircraft takes longer than expected, it is reported to the planner that handles the preflight preparation of the aircraft. That planner will seek to replan the preparation task in a way that will continue to meet its commitment to the mission planner. If it can do so, neither the mission planner nor its plan will be affected. It is only if the preflight preparation planner is unable to maintain its commitment that the mission planner needs to be invoked. The replanning activity is limited to those parts of a plan that are necessarily affected.

In the next section, we describe in detail the design of a planner and list the functions used to obtain the desired behavior.

IV. PLANNER DESIGN AND IMPLEMENTATION

In this section, the detailed design of the planners is given, including the actual functions used. The system uses INTERLISP, and the usual INTERLISP functions are assumed. In addition, the planners use some of the same functions as the schedulers of ACS.1, particularly the miscellaneous convenience functions. These have been described in Technical Report 14, "The Schedulers of ACS.1," Oct. 1977, and are not repeated here.

The organization of this section is as follows:

In A, Miscellaneous Convenience Functions, two new convenience functions are given. These are new versions of two functions that were given in Technical Report 14, which are needed in some circumstances.

In B, Initial Creation of a Process Model, the functions are given that allow the user to input the core structure of the process model, which encodes the knowledge used by a planner.

In C, Modification and Printout of a Process Model, the functions are detailed that allow a user to look at, and modify, a process model. The functions are also useful for entering additional features into the basic model created by the functions described in subsection B.

In D, Creation of a Plan, the functions used in the initial creation of a plan are described. Note that a plan exists when a structure for it exists and has been given an ID, even though none of its values has been specified. The functions described here create such a blank plan. A good deal is accomplished in this process, however, since the demons that will enforce its consistency according to the process model are created and attached to it. The details of these demons are described later, but the functions that create them are included in this subsection.

In E, Printout of a Plan, the functions that allow the user to see the current status of a given plan are given. An example of an empty plan is shown and its principal features discussed.

In F, Initial Entry of Data, the functions that allow the user to specify values for a plan are given. These include not only data to be incorporated in the plan, but also the means for the user to specify the duration of particular tasks in the plan.

In G, Local Sequential and Concurrency Demons, the demons that maintain the sequential and concurrency constraints of the model are discussed and their formats given. By a sequential constraint is meant a constraint that, for example, certain tasks must be completed before another task can be started. By a concurrency constraint is meant a constraint that, for example, a resource assignment must be concurrent with the execution of certain tasks.

In H, Local Duration Demons, the demons that maintain the durational constraints are discussed and their formats given. These constraints may be derived either from the model or from a specification of a duration entered by the user through a function described in subsection F.

In I, Type Demons, the demons that are attached either to the set of tasks as a whole, or to the set of assignments, are described. As the plan develops, these demons enforce the continuing self-consistency as a whole.

In J, Assignments and Task Planning, the mechanism that drives the planning process is given. This uses a global demon attached to the plan as a whole. It is this demon that, in ACS.1, initiates the calls on other planners or schedulers. Since attention is limited here to the planner design, the mechanism for these calls to other modules is not included. Instead, only the default condition is given, which creates a dialog with the user, permitting him to enter the subplans and assignments that may be required.

In the final subsection, K, Alert on Given Values, the local demon that watches exogenic data that have been entered into the plan is given. This demon watches these data and issues an alert message whenever they are changed in the planning process. Note that they can be changed either by the later entry of other exogenic data, or as a result of enforcing the consistency of the developing data with current time. While such a change is permitted, it is important that the user be alerted to the fact when it occurs.

Throughout the section, the actions of the various functions are illustrated in a series of tables that show the process of developing a plan.

A. Miscellaneous Convenience Functions

The planners, as described here, are intended to operate in conjunction with the schedulers described in Technical Report 14. All the functions detailed in that report are assumed to be available. In particular, the functions described there for manipulating A-lists are used here freely without description.

It has been found necessary, for reasons that have not been traced, to introduce a different version of A.REMVAL called A.REMVAL1, and a corresponding version of A.REMVAL# called A.REMVAL1#. These use PUTASSOC rather than simply adjusting the current value of X. The functions are needed to remove some rather complicated list structures from others in which they are embedded. Apparently copies are being made rather unexpectedly, and it is sometimes necessary to force the desired changes. A.REMVAL1 does this, and so is more reliable.

The two functions that need to be listed here are as follows:

```
(A.REMVAL1
  [LAMBDA (LST PROP VAL)
    (PROG (X)
          (SETQ X (ASSOC PROP LST))
          (COND
            ((NULL X)
              (RETURN NIL))
            [(EQUAL X (LIST PROP VAL))
              (COND
                ((CDR LST)
                  (A.REMPROP.S LST PROP))
                (T (RPLACA LST (LIST PROP]
            ((MEMBER VAL X)
              (PUTASSOC PROP (CDR (REMOVE VAL X))
                        LST)))
          (RETURN VAL])

(A.REMVAL1#
  [LAMBDA (LST PROP.LIST VAL)
    (COND
      ((NULL (CDR PROP.LIST))
        (A.REMVAL1 LST (CAR PROP.LIST)
                   VAL))
      (T (A.REMVAL1# (A.GETP.S LST (CAR PROP.LIST))
                     (CDR PROP.LIST)
                     VAL])
```

B. Initial Creation of a Process Model

The process models used by the planners are held in an A-list called MODELS. Each process model is the value of the property that is the name of the model. A process model, itself, is a list of four properties, each of whose values is itself an A-list; these are named TASKS, RESOURCES, TASK.SPECS and RES.SPECS. The value of TASKS is a list of the tasks that are identified as part of the process, and the value of RESOURCES is a list of the resources that are specified by the process model as needing to be assigned. Note that there may be additional resources required by other process models that decompose the tasks in a given process model. The resources listed are only those specifically named by the process model.

The value of TASK.SPECS is an A-list whose property names are, again, the names of the tasks. Their values are A-lists that describe the constraints and properties of the separate tasks. Similarly, the value of RES.SPECS is an A-list whose property names are the names of the required resources, and whose values are A-lists describing the constraints and properties of the separate resources. This structure is illustrated later.

The initial creation of a process model is done with the function P.MAKE. This constructs a simplified process model in which the tasks and resource assignments are identified and the sequential and concurrency relations between them are entered. By a sequential constraint is meant one specifying that a task must be completed before other tasks can be started. A concurrency constraint states that an assignment must be made over a period that includes the durations of certain tasks.

The skeleton process model generated by P.MAKE may require additional information identifying subplanners or schedulers or giving additional specifications such as the duration of a task. The entry of these items is discussed in the next section.

When called, P.MAKE first asks for the tasks and resource assignments needed. It then takes each task in turn and asks for a list of all immediate antecedents, or for all tasks, if any, that must be completed before the named task can be started. Then, for each resource, it asks for a list of tasks with which its assignment must be concurrent. That is, the assignment of a resource, R1, is concurrent with task T1 if the period of its assignment must include the period of T1. Note that an assignment is assumed to be continuous. If a resource must be assigned concurrently with T1 and T2, which may be separated in time, and if the assignment is not needed between the two tasks, then it should be regarded as two assignments of the same resource type.

There can be situations in which a single resource, R1, must be assigned concurrently with two tasks, T1 and T2, that may be separated in time, and in which we wish to release the resource in between. If it is necessary that R1 be the same one of a set of possible resources, additional capabilities would have to be provided in the process model and in the schedulers. We have not included the capability of specifying and handling this situation since it would introduce additional complications that are not needed in the current application.

Note also that the task and resource assignments are handled differently. No provision has been made for concurrency constraints between tasks, or for sequential constraints between assignments. Further, the concurrency constraints are assumed to require that an assignment be concurrent with one or more tasks, never with other assignments. This reflects the situation in the application environment. If required for some other application environment, additional capabilities could be included.

The actual entry of the sequential constraints among the tasks is done with P.T.MAKE. Note that, in accepting a list of antecedents of a given task, each entry is checked against the list of tasks. If not found, the user is permitted to change it, but he is not permitted to add new tasks, or to specify an antecedent that is not one of the list.

The antecedents of a given task, if any, are entered as the value of ANTECEDENTS under the task name in the A-list that is the value of TASK.SPECS in the model. If there are no antecedents of a given task, no entry is made.

After executing P.T.MAKE for all listed tasks, P.MAKE calls LATTICE to check that the sequential constraints create a partial ordering. LATTICE operates on the task specifications, TASK.SPECS, as follows: It first identifies all tasks that have no antecedents, giving them the value 0. It then identifies all tasks whose antecedents have been given the value 0, and gives them the value 1. It continues this way, assigning the next higher value to all tasks all of whose antecedents have been given values. The process terminates when either all tasks have been given values, or a pass is completed with no value being assigned. In the former case, a partial ordering exists, and the value of each task is its maximum distance from a root task, or a task with no antecedents. In the latter case, no partial ordering exists; there must be one or more cycles among the remaining tasks. If the ordering succeeds, LATTICE returns a list whose first element is T, and the succeeding terms are dotted pairs giving the value assigned to each task. If the ordering fails, the initial term of the list returned by LATTICE is NIL. The rest is a list of the tasks to which no value has been assigned. This is returned to help identify the error. P.MAKE defaults if LATTICE indicates that the ordering process failed and prints the tasks for which it failed.

At the present time, no use is made of the values assigned to the tasks by LATTICE, when it succeeds, and the values are not retained. They are returned by LATTICE, however, so that they can be entered into the model should a need be discovered at a later time.

If LATTICE succeeds, P.MAKE then calls P.COMPL. This adds to each task in the A-list that is the value of TASK.SPECS the property SUCCESSORS whose value is the list of tasks for which the given task is an antecedent. It therefore generates the complementary set of sequential constraints based on the immediate successors of each task.

Following this action, providing some resources have been named, P.MAKE asks for the concurrency constraints on the resources. Note that a concurrency constraint is not a symmetrical relation. The statement that assignment A must be concurrent with tasks T1 and T2, for example, means that the interval spanned by A must include the intervals spanned by T1 and T2. It does not preclude A spanning a greater time. In particular, it is generally sufficient to specify that A shall be concurrent with at most two tasks which can be far apart in the process model. Since assignment is assumed to be continuous, such a specification implies that A will be concurrent with all intervening tasks. Note also that concurrency is not dependent on the ordering of the tasks. If assignment A is specified to be concurrent with tasks T1 and T2, this constraint does not specify whether the start of A is controlled by the start of T1 or the start of T2, and similarly for the end. This is done to avoid requiring that the partial ordering of the tasks generate an ordering of T1 and T2, which might not be the case.

The entrance of the concurrency relations is done with the function P.R.MAKE. This function also checks that the tasks named are in the list identified before, and provides an opportunity to correct the entry if this is not true. P.R.MAKE requires that some concurrency relation be given for each assignment. Otherwise, there would be no way to relate the assignment to the rest of the process model. However, the specification of a single task is sufficient.

The list of tasks with which a given assignment must be concurrent is entered into the model as the value of CONCURRENT in the A-list that is the value of the resource name in the A-list that is the value of RES.SPECS in the model.

Once the concurrency relations have been entered, P.R.COMPL is called. This function considers each resource assignment. For each task named as a value of the property CONCURRENT, the name of the resource is added to the task specification as a value of ASSIGN, which is, then, the inverted form of the CONCURRENT property.

Finally, P.MAKE provides a printout of the process model as it exists at this time, and then offers the user the option of entering further properties, or of modifying the existing ones. The functions that do this are given in the next subsection.

To illustrate the results so far, Table 1 is a printout of the use of P.MAKE to enter the process model outlined in Figure 2.

Table 1
The Creation of a Process Model

```
(p.make 'mission)

What are the tasks in MISSION?
Enter as a list, terminated with ]:  A/C-PREP
  PILOT-BRIEF FLT A/C-SERV PILOT-DEBRIEF]

What are the resources in MISSION?
Enter as a list, terminated with ]:  A/C PILOT]

Enter first the sequential constraints among the tasks.

For A/C-SERV, what tasks must be completed first?
Enter as a list, terminated with ]:  ]

For PILOT-BRIEF, what tasks must be completed first?
Enter as a list, terminated with ]:  ]

For FLT, what tasks must be completed first?
Enter as a list, terminated with ]:  A/C-SERV
PILOT-BRIEF]

For A/C-SERV, what tasks must be completed first?
Enter as a list, terminated with ]:  FIGHT]

For PILOT-DEBRIEF, what tasks must be completed first?
Enter as a list, terminated with ]:  FLT]

Next, enter the concurrency constraints on the resources.

The assignment of A/C must be concurrent with which tasks?
Enter as a list, terminated with ]:  A/C-PREP A/C-SERV]

The assignment of PILOT must be concurrent with which tasks?
Enter as a list, terminated with ]:  PILOT-BRIEF PILOT-DEBRIEF]
```

Table 1
(Continued)

```
The model named MISSION is as follows:


Tasks:  A/C-PREP, PILOT-BRIEF, FLT, A/C-SERV,
        PILOT-DEBRIEF.
Resources:  A/C, PILOT.


Task specifications:
     A/C-PREP:
          SUCCESSORS:  (FLT)
          ASSIGN:  (A/C)
     PILOT-BRIEF:
          SUCCESSORS:  (FLT)
          ASSIGN:  (PILOT)
     FLT:
          ANTECEDENTS:  (PILOT-BRIEF, A/C-PREP)
          SUCCESSORS:  (A/C-SERV, PILOT-DEBRIEF)
     A/C-SERV:
          ANTECEDENTS:  (FLT)
          ASSIGN:  (A/C)
     PILOT-DEBRIEF:
          ANTECEDENTS:  (FLT)
          ASSIGN:  (PILOT)


Resource specifications:
     A/C:
          CONCURRENT:  (A/C-PREP, A/C-SERV)
     PILOT:
          CONCURRENT:  (PILOT-DEBRIEF, PILOT-BRIEF)

Do you wish to modify it  (Y or N)  N
OK
```

The final part of Table 1 is a printout of the model as it exists after the preceding dialog. The print function used is given later. It reformats the usual LISP printout into a more readable form. In general, parentheses are suppressed except for the values of the final properties. They have been kept there to distinguish between a list that may have only one element and an atomic value.

The functions that are used to create a process model are as follows:

```
(P.MAKE
  [LAMBDA (NAME)
    (PROG (TASK TASKLIST RESOURCE RESLIST TASK.SPECS RES.SPECS LST)
     B    (COND
            ((NULL NAME)
              (PRIN1 "What name?  ")
              (SETQ NAME (READ))
              (GO B))
            ((A.GETP MODELS NAME)
              (MAPPRINQ (TERPRI "A process model named " NAME
                                   " already exists." TERPRI))
              (RETURN NIL)))
          (MAPPRINQ (TERPRI "What are the tasks in " NAME "?" TERPRI
                            "Enter as a list, terminated with ]:  "))
     L    (COND
            ((SETQ TASK (READ))
              (SETQ TASKLIST (CONS TASK TASKLIST))
              (GO L)))
          (SETQ TASKLIST (REVERSE TASKLIST))
          (MAPPRINQ (TERPRI "What are the resources in " NAME "?"
                            TERPRI
                            "Enter as a list, terminated with ]:  "))
     L1   (COND
            ((SETQ RESOURCE (READ))
              (SETQ RESLIST (CONS RESOURCE RESLIST))
              (GO L1)))
          (MAPPRINQ (TERPRI
              "Enter first the sequential constraints among the tasks."
                            TERPRI))
          [SETQ TASK.SPECS (MAPCONC (COPY TASKLIST)
                                    (FUNCTION (LAMBDA (X)
                                          (LIST (LIST X]
          [MAPC TASKLIST (FUNCTION (LAMBDA (X)
                    (P.T.MAKE X TASKLIST TASK.SPECS]
          [COND
            ((CAR (SETQ LST (LATTICE TASK.SPECS)))
              (P.COMPL TASK.SPECS))
            (T (PRIN1 "Tasks cannot be ordered.")
               (TERPRI)
               (MAPRINT (CDR LST)
                        T "The tasks that fail are:  " (QUOTE %.)
                        ", ")
               (RETURN (CHARACTER 127]
            (A.PUT MODELS NAME (LIST (CONS (QUOTE TASKS)
                                           TASKLIST)))
            (A.PUT# MODELS (LIST NAME (QUOTE TASK.SPECS))
                    TASK.SPECS)
            (COND
              (RESLIST (SETQ RESLIST (REVERSE RESLIST)))
              (T (GO FINAL)))
            (MAPPRINQ (TERPRI
              "Next, enter the concurrency constraints on the resources."
                                TERPRI))
            [SETQ RES.SPECS (MAPCONC (COPY RESLIST)
                                     (FUNCTION (LAMBDA (X)
                                         (LIST (LIST X]
            [MAPC RESLIST (FUNCTION (LAMBDA (X)
                        (P.R.MAKE X TASKLIST RES.SPECS]
            (A.PUT# MODELS (LIST NAME (QUOTE RESOURCES))
                    RESLIST)
            (A.PUT# MODELS (LIST NAME (QUOTE RES.SPECS))
                    RES.SPECS)
            (P.R.COMPL RES.SPECS TASK.SPECS)
        FINAL
            (TERPRI)
            [MAPPRINQ ("The model named " NAME " is as follows:"
                                              (RPTQ 2 (TERPRI]
            (P.PRINT NAME)
            (PRIN1 "Do you wish to modify it?  (Y or ])  ")
            (COND
              ((EQUAL (READ)
                      (QUOTE Y))
                (P.MODIFY NAME)
                (GO FINAL))
              (T (PRIN1 "OK")
                 (RETURN (CHARACTER 127])

  (P.T.MAKE
    [LAMBDA (TASK TASKLIST TASK.SPECS)
      (PROG (TRIAL LST ENTRY)
            (MAPPRINQ (TERPRI "For " TASK
                              ", what tasks must be completed first?"
                              TERPRI
                              "Enter as a list, terminated with ]:  "))
        L   (COND
              ((SETQ TRIAL (READ))
                (SETQ LST (CONS TRIAL LST))
                (GO L)))
        L1  [COND
              ((NULL LST)
                (RETURN NIL))
              (T (SETQ TRIAL (CAR LST))
                 (SETQ LST (CDR LST]
L2    [COND
        [(MEMBER TRIAL TASKLIST)
          (COND
            ((SETQ ENTRY (A.GETP TASK.SPECS TASK))
              (A.ADDPROP ENTRY (QUOTE ANTECEDENTS)
                           TRIAL))
            (T (A.ADDPROP TASK.SPECS TASK (LIST (QUOTE ANTECEDENTS)
                                                  TRIAL]
        (T (MAPPRINQ (TRIAL
                   " not listed as a task.  What task was meant?"))
                      TERPRI
               "Enter correct name or ] to cancel entry:  "))
           (COND
             ((SETQ TRIAL (READ))
               (GO L2))
             (T (GO L1]
      (GO L1])

(LATTICE
  [LAMBDA (TASK.SPECS)
    (PROG (LST ORDER.LIST TEST.LIST (FLG T)
           (N 1)
           T.LIST)
      (SETQ LST (COPY TASK.SPECS))
      [MAPC (COPY TASK.SPECS)
            (FUNCTION (LAMBDA (X)
                 (COND
                   ((NULL (A.GETP X (QUOTE ANTECEDENTS)))
                     (SETQ ORDER.LIST (CONS (CONS (CAR X)
                                                  0)
                                            ORDER.LIST))
                     (SETQ TEST.LIST (CONS (CAR X)
                                           TEST.LIST))
                     (COND
                       ((CDR LST)
                         (SETQ LST (REMOVE X LST)))
                       (T (RETURN (CONS T ORDER.LIST]
L     [COND
        (FLG)
        (LST (RETURN (CONS NIL LST)))
        (T (RETURN (CONS T ORDER.LIST]
      (SETQ FLG NIL)
      (SETQ T.LIST (COPY TEST.LIST))
```

```
          [MAPC (COPY LST)
                (FUNCTION (LAMBDA (X)
                    (COND
                      ([EVERY (A.GETP X (QUOTE ANTECEDENTS))
                              (FUNCTION (LAMBDA (Y)
                                  (MEMBER Y T.LIST]
                        (SETQ ORDER.LIST (CONS (CONS (CAR X)
                                                     N)
                                               ORDER.LIST))
                        (SETQ TEST.LIST (CONS (CAR X)
                                              TEST.LIST))
                        (SETQ FLG T)
                        (COND
                          ((CDR LST)
                            (SETQ LST (REMOVE X LST)))
                          (T (RETURN (CONS T ORDER.LIST]
          (SETQ N (ADD1 N))
          (GO L])

(P.COMPL
  [LAMBDA (TASK.SPECS)
    (MAPC TASK.SPECS (FUNCTION (LAMBDA (X)
              (PROG (LST.1 LST.2)
                    [SETQ LST.1 (SUBSET TASK.SPECS
                                        (FUNCTION (LAMBDA (Y)
                                           (AND
                                             (SETQ LST.2
                                               (A.GETP Y
                                                 (QUOTE ANTECEDENTS)))
                                             (MEMBER (CAR X)
                                                     LST.2]
                    (COND
                      [LST.1 (MAPC LST.1 (FUNCTION (LAMBDA (Z)
                                       (A.ADDPROP X (QUOTE SUCCESSORS)
                                                  (CAR Z]
                      (T (RETURN NIL])
(P.R.MAKE
  [LAMBDA (RESOURCE TASKLIST RES.SPECS)
    (PROG (TRIAL LST LST.1 ENTRY)
      B   (MAPPRINQ (TERPRI "The assignment of " RESOURCE
                            " must be concurrent with which tasks?"
                            TERPRI
                            "Enter as a list, terminated with ]:  "))
      L   (COND
            ((SETQ TRIAL (READ))
              (SETQ LST (CONS TRIAL LST))
              (GO L)))
      L3  (COND
            ((NULL LST)
              (MAPPRINQ (TERPRI "Some concurrency relation is needed."
                                TERPRI))
              (GO B)))
          (SETQ LST.1 (COPY LST))
      L1  [COND
            ((NULL LST)
              (RETURN NIL))
            (T (SETQ TRIAL (CAR LST))
               (SETQ LST (CDR LST]
      L2  [COND
            [(MEMBER TRIAL TASKLIST)
              (COND
                ((SETQ ENTRY (A.GETP RES.SPECS RESOURCE))
                  (A.ADDPROP ENTRY (QUOTE CONCURRENT)
                             TRIAL))
                (T (A.ADDPROP RES.SPECS RESOURCE (LIST
                                                   (QUOTE CONCURRENT)
                                                   TRIAL]
            (T (MAPPRINQ (TRIAL
                          " not a listed task.  What task was meant?"
                               TERPRI
                               "Enter correct name or ] to cancel:  "))
               (COND
                 ((SETQ TRIAL (READ))
                   (GO L2))
                 ((CDR LST.1)
                   (GO L1))
                 (T (GO L3]
          (GO L1])
```

```
(P.R.COMPL
  [LAMBDA (RES.SPECS TASK.SPECS)
    (MAPC RES.SPECS (FUNCTION (LAMBDA (X)
                 (PROG (LST)
                       (COND
                         ((SETQ LST (A.GETP X (QUOTE CONCURRENT)))
                           (MAPC LST (FUNCTION (LAMBDA (Y)
                                       (A.ADDPROP# TASK.SPECS
                                                   (LIST Y (QUOTE ASSIGN))
                                                   (CAR X])
```

C. Modification and Printout of a Process Model

Once a process model is in existence, it can be displayed conveniently by P.PRINT. P.PRINT calls P.PRINT.1 for the TASK.SPECS and RES.SPECS components. P.PRINT.1, in turn, calls P.PRINT.2 for each component--i.e., for each task or resource. P.PRINT is called by P.MAKE, leading to the printout of the model shown at the end of Table 1.

These special print functions have been written, rather than using the more general PRINT.LIST given in Technical Report 14, since there are values that are properly lists and should be printed as such.

P.MAKE, short of the final optional call on P.MODIFY, creates what might be called the skeleton of a process model. It identifies the tasks and resources and enters the sequential and concurrency constraints among them. There are many other properties and constraints that may be needed for them. These are entered by P.MODIFY, which also permits modification of existing properties.

P.MODIFY sorts out whether the user wants to modify the task or resource specifications. It also permits the user to enter a new section called MODEL.SPECS, or to modify it once it has been created. MODEL.SPECS is used to hold any constraints or requirements that are not associated with either the tasks or the resources, but are global to the entire model. For example, there may be need of rules that determine the order of implementation of the model into a plan. This may be desirable to avoid excessive replanning. Either the rule itself would be encoded here, or the name of a function that, when called, will determine the next task or resource assignment to be sought.

Note that it is possible to use P.MODIFY to change the sequential or concurrency relations. It is dangerous to do so, however, since no check is made that internal consistency is maintained. If a major change is to be made, it is probably better to delete the model and recreate it.

Note also that there is no provision for entering new tasks or resources, or for deleting old ones. Where this is necessary, it is, again, generally better to delete the model and recreate it, so that consistency can be assured.

P.MAKE calls MODIFY.T.SPECS, MODIFY.R.SPECS, or MODIFY.M.SPECS, depending on the part of the process model indicated by the user. At the end of each, the user can reenter the function to make additional changes or additions. If he does not want to do this, control reverts to P.MODIFY and he can designate a different component for attention.

The functions given provide convenient means for constructing or modifying a process model.

To avoid any possible confusion from changing a process model "on the fly," the function RENAME.P is defined. It creates a new copy of an existing model with a new name, which can then be modified as desired leaving the old version intact. Note that it does actually make a new copy before entering it into MODELS.

The functions that permit the user to modify an existing process model, and to print it in convenient format, are as follows:

```
(P.PRINT
  [LAMBDA (NAME)
    (PROG (MODEL RESOURCES RES.SPECS MODEL.SPECS)
          (COND
            ((SETQ MODEL (A.GETP MODELS NAME)))
            (T (MAPPRINQ ("No model named " NAME " listed." TERPRI))
              (RETURN NIL)))
          (TERPRI)
          (MAPRINT (A.GETP MODEL (QUOTE TASKS))
                   T "Tasks:  " (QUOTE %.)
                   ", ")
          (TERPRI)
          (COND
            ((SETQ RESOURCES (A.GETP MODEL (QUOTE RESOURCES)))
              (MAPRINT RESOURCES T "Resources:  " (QUOTE %.)
                       ", "))
            (T (PRIN1 "Resources:  none specified.")))
          (TERPRI)
          (RPTQ 2 (TERPRI))
          (PRIN1 "Task specifications:  ")
          (TERPRI)
          (P.PRINT.1 (A.GETP MODEL (QUOTE TASK.SPECS)))
          (COND
            ((SETQ RES.SPECS (A.GETP MODEL (QUOTE RES.SPECS)))
              (RPTQ 2 (TERPRI))
              (PRIN1 "Resource specifications:  ")
              (TERPRI)
              (P.PRINT.1 RES.SPECS)
              (TERPRI)))
          (COND
            ((SETQ MODEL.SPECS (A.GETP MODEL (QUOTE MODEL.SPECS)))
              (RPTQ 2 (TERPRI))
              (PRIN1 "Model specifications:")
              (TERPRI)
              (P.PRINT.2 MODEL.SPECS)
              (TERPRI)))
          (RETURN (CHARACTER 127])

(P.PRINT.1
  [LAMBDA (LST)
    (MAPC LST (FUNCTION (LAMBDA (X)
              (PROG (VAL)
                    (SETQ VAL (CDR X))
                    (MAPPRINQ ((TAB 5)
                               (CAR X)
                               ":  "))
                    (COND
                      ((NULL VAL)
                        (PRIN1 "not specified.")
                        (TERPRI))
                      ((ATOM VAL)
                        (MAPPRINQ (VAL "." TERPRI)))
                      (T (P.PRINT.2 VAL])

(P.PRINT.2
  [LAMBDA (LST)
    (MAPC LST (FUNCTION (LAMBDA (X)
              (COND
                ((NULL (CDR X))
                  (MAPPRINQ ((TAB 10)
                             (CAR X)
                             ":  not specified." TERPRI)))
                ((ATOM (CDR X))
                  (MAPPRINQ ((TAB 10)
                             (CAR X)
                             ":  "
                             (CDR X)
                             TERPRI)))
                (T (MAPPRINQ ((TAB 10)
                              (CAR X)
                              ":  "))
                   (MAPRINT (CDR X)
                            T NIL (QUOTE %.) ", ")
                   (TERPRI])
```

```
(P.MODIFY
  [LAMBDA (NAME)
    (PROG (MODEL X)
          (COND
            ((SETQ MODEL (A.GETP MODELS NAME)))
            (T (MAPPRINQ ("No model named " NAME " listed." TERPRI))
               (RETURN NIL)))
      L   (MAPPRINQ (TERPRI
           "Do you wish to modify the task, resource, or model specs?"
                             TERPRI
                      "Enter TASKS, RESOURCES, MODEL or ] to abort:  "))
          (COND
            ((NULL (SETQ X (READ)))
              (PRIN1 "OK")
              (RETURN (CHARACTER 127)))
            ((EQUAL X (QUOTE TASKS))
              (MODIFY.T.SPECS MODEL))
            ((EQUAL X (QUOTE RESOURCES))
              (MODIFY.R.SPECS MODEL))
            ((EQUAL X (QUOTE MODEL))
              (MODIFY.M.SPECS MODEL))
            (T (PRIN1 "Do not understand.")
               (TERPRI)
               (GO L)))
          (MAPPRINQ (TERPRI
                        "Do you want to see the model?  (Y or ]):  "))
          (COND
            ((EQUAL (READ)
                    (QUOTE Y))
              (P.PRINT NAME)))
          (MAPPRINQ (TERPRI
           "Do you want to make further modifications?  (Y or ]):  "))
          (COND
            ((EQUAL (READ)
                    (QUOTE Y))
              (GO L))
            (T (PRIN1 "OK")
               (RETURN (CHARACTER 127])
(MODIFY.T.SPECS
  [LAMBDA (MODEL)
    (PROG (TASK PROP PROP.LIST)
      L    (MAPPRINQ (TERPRI
                "Which task do you want to modify?  (One only):  "))
           (SETQ TASK (READ))
      L1   (COND
             ((NULL TASK)
               (RETURN NIL))
             [(MEMBER TASK (A.GETP MODEL (QUOTE TASKS]
             (T (MAPPRINQ (TASK " is not one of the tasks in the model."
                                TERPRI
                                "Re-enter it, please, or ] to abort:  "))
                (SETQ TASK (READ))
                (GO L1)))
           (MAPPRINQ (TERPRI "What properties of " TASK
                             " are to be added, modified or deleted?"
                             TERPRI
                             "Enter as a list, terminated with ]:  "))
      L2   (COND
             ((SETQ PROP (READ))
               (SETQ PROP.LIST (CONS PROP PROP.LIST))
               (GO L2)))
           [COND
             (PROP.LIST (MAPC PROP.LIST
                              (FUNCTION (LAMBDA (X)
                                  (PROG (VAL)
                                        (MAPPRINQ (TERPRI (TAB 5)
                                      "What value should be entered for "
                                                           X "?  " TERPRI
                                                           (TAB 5)
                                        "Enter ] to delete property:  "))
                                        (COND
                                          ((SETQ VAL (READ))
                                            (A.PUT# MODEL
                                                    (LIST
                                                      (QUOTE TASK.SPECS)
                                                      TASK X)
                                                    VAL))
                                          (T (A.REMPROP# MODEL
                                                         (LIST
                                                           (QUOTE
                                                             TASK.SPECS)
                                                           TASK X]
           (MAPPRINQ (TERPRI
                 "Is there another task to be modified?  (Y or ]):  "))
```

```
         (COND
           ((EQUAL (READ)
                   (QUOTE Y))
             (SETQ PROP.LIST NIL)
             (GO L))
           (T (RETURN NIL])

(MODIFY.R.SPECS
  [LAMBDA (MODEL)
    (PROG (R.LIST RESOURCE PROP PROP.LIST)
          (COND
            [(SETQ R.LIST (A.GETP MODEL (QUOTE RESOURCES]
            (T (PRIN1 "No resources listed.")
               (RETURN NIL)))
      L   (MAPPRINQ (TERPRI
               "Which resource do you want to modify?  (One only):  "))
          (SETQ RESOURCE (READ))
      L1  (COND
            ((NULL RESOURCE)
              (RETURN NIL))
            ((MEMBER RESOURCE R.LIST))
            (T (MAPPRINQ (TERPRI RESOURCE
                           " is not one of the resources of the model."
                                 TERPRI
                              "Re-enter it, please, or ] to abort:  "))
               (SETQ RESOURCE (READ))
               (GO L1)))
          (MAPPRINQ (TERPRI
              "What properties are to be added, modified, or deleted?"
                             TERPRI
                             "Enter as a list, terminated with ]:  "))
      L2  (COND
            ((SETQ PROP (READ))
              (SETQ PROP.LIST (CONS PROP PROP.LIST))
              (GO L2)))
```

```
        [COND
          (PROP.LIST (MAPC (REVERSE PROP.LIST)
                           (FUNCTION (LAMBDA (X)
                               (PROG (VAL)
                                     (MAPPRINQ (TERPRI (TAB 5)
                                   "What value should be entered for "
                                                       X "?" TERPRI
                                                       (TAB 5)
                                     "Enter ] to delete property:  "))
                                     (COND
                                       ((SETQ VAL (READ))
                                         (A.PUT# MODEL
                                                (LIST
                                                  (QUOTE RES.SPECS)
                                                  RESOURCE X)
                                                VAL))
                                       (T (A.REMPROP#
                                            MODEL (LIST
                                                    (QUOTE RES.SPECS)
                                                    RESOURCE X]
        (MAPPRINQ (TERPRI
           "Is there another resource to be modified?  (Y or ]):  "))
        (COND
          ((EQUAL (READ)
                  (QUOTE Y))
            (SETQ PROP.LIST NIL)
            (GO L))
          (T (RETURN NIL])

(MODIFY.M.SPECS
  [LAMBDA (MODEL)
    (PROG (PROP PROP.LIST)
      L   (MAPPRINQ (TERPRI
         "What model properties are to be added, deleted or modified?"
                          TERPRI
                          "Enter as a list, terminated with ]:  "))
      L1  (COND
            ((SETQ PROP (READ))
              (SETQ PROP.LIST (CONS PROP PROP.LIST))
              (GO L1)))
```

```
[COND
  (PROP.LIST (MAPC (REVERSE PROP.LIST)
                   (FUNCTION (LAMBDA (X)
                       (PROG (VAL)
                             (MAPPRINQ (TERPRI (TAB 5)
                         "What value should be entered for "
                                                X "?" TERPRI
                                                (TAB 5)
                                          "Enter ] to delete "
                                                X ".  "))
                             (COND
                               ((NULL (SETQ VAL (READ)))
                                 (A.REMPROP#
                                   MODEL
                                   (LIST (QUOTE MODEL.SPECS)
                                         X)))
                               ((NULL (A.GETP
                                        MODEL (QUOTE
                                                MODEL.SPECS)))
                                 (A.ADDPROP MODEL
                                           (QUOTE
                                             MODEL.SPECS)
                                           (CONS X VAL)))
                               (T (A.PUT# MODEL
                                         (LIST
                                           (QUOTE
                                             MODEL.SPECS)
                                           X)
                                         VAL]
  (MAPPRINQ (TERPRI
"Any other changes to the model specifications?  (Y or ]):  "))
  (COND
    ((EQUAL (READ)
            (QUOTE Y))
      (SETQ PROP.LIST NIL)
      (GO L)))
  (RETURN NIL])
```

```
(RENAME.P
  [LAMBDA (OLD.NAME NEW.NAME)
    (PROG (MODEL)
          (COND
            ((SETQ MODEL (A.GETP MODELS OLD.NAME)))
            (T (MAPPRINQ ("No model named " OLD.NAME " listed." TERPRI))
               (RETURN NIL)))
      L   (COND
            ((NULL NEW.NAME)
              (PRIN1 "What is new name?")
              (SETQ NEW.NAME (READ))
              (GO L))
            ((A.GETP MODELS NEW.NAME)
              (MAPPRINQ (NEW.NAME " already names a model.  No entry."
                                   TERPRI))
              (RETURN NIL)))
          (A.PUT MODELS NEW.NAME (COPY MODEL))
          (PRIN1 "OK")
          (RETURN (CHARACTER 127])
```

D. Creation of a Plan

By a "plan" is meant a data structure that, when filled, will describe how a process model is expected to be instantiated. A plan is said to exist whether or not the planning procedure has been completed, or even started. The initial creation of a plan generates the required data structure and gives it a label. The data structure is an active one. It is set up so that it will do three things:

* Specify, by implication, what data must exist for the plan to be complete

* Enforce automatically the sequential and concurrency relations of the model

* Drive the planning or replanning process as far as possible, given the data that it contains.

The first property is obtained by establishing special substructures that contain slots for all data that may be present, and filling the slots with "T" if they are required. The plan is complete only when none of these slots have the value T. The other properties are obtained by a rather complex system of demons whose detailed structure and operation is discussed in later sections.

Plans are contained in a larger data structure called PLANS which is initialized to (DEFAULT). They are contained in this as a set of A-lists, which are the values of properties whose names are the corresponding process models. An individual plan is contained in the appropriate A-list as the value of the ID of the plan, and is itself an A-list. *

The structure of a plan is, again, an A-list. It has two principal components, TASKS and RESOURCES. In addition, there may be a component whose name is DEMONS and whose value is a list of what we call the "global demons," which operate on the plan as a whole. The value of TASKS is an A-list whose primary components are the tasks,

----

* An A-list, for "association list," is a list of property-value pairs. The order of the pairs is not specified and retrieval is always by the name of the desired property, or is associative.

The value of a property in an A-list may be an A-list itself, so that the definition is recursive, and the complete structure of an A-list can be a tree of any desired depth.

considered as property names. In addition, there may be four other components, called DEMONS, PLANNABLE, UNPLANNED, and DEFERRED. The value of DEMONS is a list of what we call the "type demons" of the task component of the plan, which may be fired when the TASKS component of the plan is modified. The value of PLANNABLE is a list of tasks for which there is enough data to permit its planning to proceed. The value of UNPLANNED is the list of tasks that have not yet been planned, set initially to the complete list of tasks. The value of DEFERRED is a list of tasks for which the planning has been deferred, using an estimated duration instead, as is discussed later. It is set initially to NIL.

The value of RESOURCES is a similar A-list whose primary components are the resources. It may also have additional components labelled DEMONS, PLANNABLE, and UNPLANNED.

In the component of TASKS under a given task name, the primary one is called TASK.PLAN. This identifies the data that must be present for a plan to be complete by initially setting it to T. In ACS.1, this is done for the start and end time of each task. There is the possibility that one might not wish to carry the planning to completion. A contingency plan, for example, might leave all times unspecified, and be considered complete when it is assured that it can be fully instantiated when needed. Also, it might not be required that all the subactivities be planned in detail during the planning. Some activities might be left undetermined on the basis that it will be possible to fill them in when needed. Leaving them unspecified can avoid the generation of apparent resource conflicts that are not substantive. It is to make possible the handling of such situations that the plan, in its initial state, is made to specify what values must be filled in for the plan to be complete.

The component of TASKS under a given task name may also include entries with the names UNPLANNED.ANTECEDENTS, UNPLANNED.SUCCESSORS, UNPLANNED.ASSIGN, DURATION, EST.DUR (for estimated duration), EST (for earliest start time), LST (for latest start time), EET (for earliest end time), LET (for latest end time), and DEMONS.

The lists of tasks, if any, under UNPLANNED.ANTECEDENTS and UNPLANNED.SUCCESSORS are used to avoid the premature planning of a task. For example, if a task has two successors, one of which has been planned, it would be possible to plan the task. However, subsequent planning of the second successor would be likely to create a conflict, forcing the task to be replanned. Therefore, a task with any unplanned successors (or antecedents) should not be chosen as the next task to be planned if there is any better choice.

The list of resource assignments, if any, which is the value of UNPLANNED.ASSIGN, is not currently used, but is retained against the possibility that we might need it at some future time.

The values of EST, LST, EET and LET, if present, translate the sequential and concurrency relations into concrete terms. The use of these is discussed later. As is also discussed later, the values that occur for these properties are not simply numbers, but include additional information.

Under the conditions that have been assumed for the application environment, the tasks will contain, at most, values for LST and EET, while the resources may contain values only for EST and LET. This fact depends on two assumptions. The first, as stated before, is that the sequential constraints are between tasks, and the concurrency constraints relate a resource assignment to a task. The second assumption is that all data are entered as a start or end time of a task, rather than of a resource. We have retained the possibility for all four properties on either a task or a resource, in order to maintain flexibility.

The values for DURATION or EST.DUR may be obtained from the model, or may be entered directly by the user. The latter case is discussed later. It is important to note, however, that if the model specifies either a duration or an estimated one, and the user later enters a different duration, the user's entry will control planning. It is for this reason that the value of the duration is transferred to the plan from the model. It makes the value available for user modification in a particular plan.

The demons included under a named task in TASKS are called the local type demons for TASKS. They may be fired when an entry is made to the TASK.PLAN of the task.

The primary components of RESOURCES, those listed under a given resource name, are similarly constructed. The component that specifies the essential data is called RES.PLAN. In ACS.1, it lists the name, start time, and end time. The auxiliary components are called UNPLANNED.CONCUR, EST, LST, EET, LET, and DEMONS.

The function that creates an empty plan--i.e., does not fill any data in--is MAKE.PLAN. It can be called with or without specifying an ID. Called from the top level, in response to a user's request for a plan, it is generally called without an ID. However, it may also be called from another planner that needs one of its tasks planned. In this case, the ID of the larger plan is passed to it.

If called without an ID, MAKE.PLAN uses GET.ID to obtain a new ID. This consults a list, FREE.ID, which is initially (1). It returns the first member of the FREE.ID list. If FREE.ID is more than one member long, the first member is removed; if it is only one member long, it is changed to a list of one element whose value is increased by one from what it was. If an ID is later released, it is added to the FREE.ID list. Therefore, IDs can be reused after they have served their original purpose.

MAKE.PLAN also makes an entry into a data structure called ID.LIST, initialized to ((0)). This is regarded as an A-list, with the IDs currently in use being its property names, except for 0, which is a default entry, present to make sure that there is never need to reduce the list to an empty list. The value of an ID in ID.LIST is a list of the process model names for which plans have been created with the given ID. When an ID is removed from current use, this list indicates where the plans are that should also be deleted.

MAKE.PLAN also calls MAKE.PLAN.DEMONS, which creates the demons needed to meet the last two objectives--i.e., to enforce the sequential and concurrency constraints, and to drive the planning process. The effect of this function, and its details, are discussed later, when we consider what demons are necessary and their structure.

The printout of a plan, permitting one to determine its current state of instantiation, is discussed in the next subsection.

The functions described in this section are defined as follows:

```
(MAKE.PLAN
  [LAMBDA (NAME ID PRINT.SUP.FLG)
    (PROG (TASKLIST TASK.SPECS LST.1 LST.2 RESOURCELIST DURATION
                    RES.SPECS)
          (COND
            [(SETQ TASKLIST (A.GETP# MODELS (LIST NAME (QUOTE TASKS]
            (T (MAPPRINQ ("Do not recognize " NAME
                                              " as a process model."
                                              TERPRI))
               (RETURN NIL)))
          (COND
            ((NULL ID)
              (SETQ ID (GET.ID NAME)))
            ((A.GETP.S# PLANS (LIST NAME ID))
              (MAPPRINQ (ID " already in use.  No entry." TERPRI))
              (RETURN NIL))
            (T (A.ADDPROP.S ID.LIST ID NAME)))
          [SETQ TASK.SPECS (A.GETP# MODELS (LIST NAME
                                                 (QUOTE TASK.SPECS]
[MAPC TASKLIST (FUNCTION (LAMBDA (X)
         (PROG (L1 SUCCESSORS ANTECEDENTS ASSIGN)
               [SETQ L1 (LIST X (COPY (QUOTE (TASK.PLAN
                                                (START . T)
                                                (END . T]
               (COND
                 ([SETQ DURATION (A.GETP# TASK.SPECS
                                          (LIST X
                                            (QUOTE DURATION]
                   (A.PUT L1 (QUOTE DURATION)
                          DURATION))
                 ([SETQ DURATION (A.GETP# TASK.SPECS
                                          (LIST X
                                            (QUOTE EST.DUR]
                   (A.PUT L1 (QUOTE EST.DUR)
                          DURATION)))
               (COND
                 ([SETQ SUCCESSORS
                     (COPY (A.GETP# TASK.SPECS (LIST X
                                                  (QUOTE
                                                  SUCCESSORS]
                   (A.PUT L1 (QUOTE UNPLANNED.SUCCESSORS)
                          SUCCESSORS)))
               (COND
                 ([SETQ ANTECEDENTS
                     (COPY (A.GETP# TASK.SPECS (LIST X
                                                  (QUOTE
                                                 ANTECEDENTS]
                   (A.PUT L1 (QUOTE UNPLANNED.ANTECEDENTS)
                          ANTECEDENTS)))
               (COND
                 ([SETQ ASSIGN (COPY (A.GETP# TASK.SPECS
                                              (LIST X
                                                (QUOTE
                                                 ASSIGN]
                   (A.PUT L1 (QUOTE UNPLANNED.ASSIGN)
                          ASSIGN)))
               (COND
                 (LST.1 (SETQ LST.1 (CONS L1 LST.1)))
                 (T (SETQ LST.1 (LIST L1]
[COND
  [(A.GETP PLANS NAME)
    (A.ADDPROP# PLANS (LIST NAME ID)
               (CONS (QUOTE TASKS)
                     (REVERSE LST.1]
  (T (A.ADDPROP PLANS NAME (LIST ID (CONS (QUOTE TASKS)
                                         (REVERSE LST.1]
(COND
  ([SETQ RESOURCELIST (A.GETP# MODELS (LIST NAME (QUOTE
                                                   RESOURCES]
    [SETQ RES.SPECS (A.GETP# MODELS (LIST NAME (QUOTE
                                                 RES.SPECS]
    [MAPC RESOURCELIST
          (FUNCTION (LAMBDA (X)
              (PROG (L1 CONCURRENT)
                    [SETQ L1
                      (LIST X (COPY (QUOTE (RES.PLAN
                                              (NAME . T)
                                              (START . T)
                                              (END . T]
                    (COND
                      ([SETQ CONCURRENT
                           (COPY (A.GETP# RES.SPECS
                                          (LIST X (QUOTE
                                                   CONCURRENT]
                         (A.PUT L1 (QUOTE UNPLANNED.CONCUR)
                                CONCURRENT)))
                    (COND
                      (LST.2 (SETQ LST.2 (CONS L1 LST.2)))
                      (T (SETQ LST.2 (LIST L1]
    (A.PUT.S# PLANS (LIST NAME ID (QUOTE RESOURCES))
              (REVERSE LST.2))
    (A.PUT.S# PLANS (LIST NAME ID (QUOTE RESOURCES)
                          (QUOTE UNPLANNED))
              RESOURCELIST)))
(A.PUT.S# PLANS (LIST NAME ID (QUOTE TASKS)
                      (QUOTE UNPLANNED))
          TASKLIST)
(MAKE.PLAN.DEMONS NAME ID)
(COND
  (PRINT.SUP.FLG (RETURN T))
  (T (MAPPRINQ ("Empty plan constructed with ID " ID "."))
     (RETURN (CHARACTER 127])
(MAKE.PLAN.DEMONS
  [LAMBDA (MODEL.NAME ID)
    (PROG (MODEL PLAN RESOURCES)
          (COND
            ((SETQ MODEL (A.GETP MODELS MODEL.NAME)))
            (T (MAPPRINQ ("No model named " MODEL.NAME "." TERPRI))
               (RETURN NIL)))
          (COND
            [(SETQ PLAN (A.GETP.S# PLANS (LIST MODEL.NAME ID]
            (T (MAPPRINQ ("No plan with ID " ID " for model named "
                                                MODEL.NAME "." TERPRI))
               (RETURN NIL)))
          (SET.SEQ.DEMONS MODEL.NAME ID PLAN (A.GETP MODEL
                                                      (QUOTE TASKS))
                          (A.GETP MODEL (QUOTE TASK.SPECS)))
          (COND
            ((SETQ RESOURCES (A.GETP MODEL (QUOTE RESOURCES)))
              (SET.CONCUR.DEMONS MODEL.NAME ID PLAN RESOURCES
                                 (A.GETP MODEL (QUOTE RES.SPECS)))
              (SET.RES.DEMONS MODEL.NAME ID PLAN)))
          (SET.DURATION.DEMONS MODEL.NAME ID MODEL PLAN RESOURCES)
          (SET.TASK.DEMONS MODEL.NAME ID PLAN)
          (SET.P.DEMONS MODEL.NAME ID PLAN])

(GET.ID
  [LAMBDA (SOURCE)
    (PROG (ID)
          (SETQ ID (CAR FREE.ID))
          [COND
            ((CDR FREE.ID)
              (SETQ FREE.ID (CDR FREE.ID)))
            (T (SETQ FREE.ID (LIST (ADD1 ID]
          (A.ADDPROP.S ID.LIST ID SOURCE)
          (RETURN ID])
```

```
(GET.ID
  [LAMBDA (SOURCE)
    (PROG (ID)
          (SETQ ID (CAR FREE.ID))
          [COND
            ((CDR FREE.ID)
              (SETQ FREE.ID (CDR FREE.ID)))
            (T (SETQ FREE.ID (LIST (ADD1 ID]
          (A.ADDPROP.S ID.LIST ID SOURCE)
          (RETURN ID])
```

E. Printout of a Plan

Once a plan exists, whether data has been entered into it or not, is printed out by PRINT.PLAN which calls PRINT.PLAN.1, PRINT.PLAN.2 and, if required, PRINT.DEMONS.

PRINT.PLAN requires two arguments, the name of the process model and the ID of the plan. It defaults if either is missing or incorrect. In addition, there are two flags that can be set, called STATUS.FLG and DEMON.FLG. If neither is set, it prints only the essential information of the plan--the start and end times of the tasks and the name, start and end times of the resource assignments. If these properties are required, it prints T for each. As discussed later, when values are given for the start and end times, they are not simple numbers but have other information associated with them. PRINT.PLAN.2 prints this additional information. An example is given later in Table 4.

If STATUS.FLG is set, it prints the additional information that is contained in a plan, such as what limits have been set for the tasks and resources, what tasks and resources are still unplanned, which of them are plannable, and so on.

If DEMON.FLG is set, the demons are printed out. If the process model is not trivial, a plan for it is likely to have many demons. The full printout of the plan, including its demons, can be quite lengthy. Some caution is advisable in calling PRINT.PLAN with DEMON.FLG set.

To illustrate these functions, Table 2a shows the printout obtained by calling PRINT.PLAN, without either flag set, immediately after the generation of a plan by MAKE.PLAN. Table 2b shows the result if STATUS.FLG is set.

TABLE 2
Printout of a Plan for MISSION

2a
Without STATUS.FLG set

```
(print.plan 'mission 1]
Plan number 1 of the model named MISSION is as follows:

Tasks:
     A/C-PREP:
          Start:  T.
          End:  T.
     PILOT-BRIEF:
          Start:  T.
          End:  T.
     FLT:
          Start:  T.
          End:  T.
     A/C-SERV:
          Start:  T.
          End:  T.
     PILOT-DEBRIEF:
          Start:  T.
          End:  T.

Resources:
     A/C:
          Name:  T.
          Start:  T.
          End:  T.
     PILOT:
          Name:  T.
          Start:  T.
          End:  T.
```

Table 2b
Printout of a Plan
With STATUS.FLG set

```
Tasks:
     A/C-PREP:
          Start:  T.
          End:  T.
          EST.DUR:  90.
          UNPLANNED.SUCCESSORS:  (FLT).
          UNPLANNED.ASSIGN:  (A/C).
     PILOT-BRIEF:
          Start:  T.
          End:  T.
          DURATION:  120.
          UNPLANNED.SUCCESSORS:  (FLT).
          UNPLANNED.ASSIGN:  (PILOT).
     FLT:
          Start:  T.
          End:  T.
          UNPLANNED.SUCCESSORS:  (A/C-SERV PILOT-DEBRIEF).
          UNPLANNED.ANTECEDENTS:  (PILOT-BRIEF A/C-PREP).
     A/C-SERV:
          Start:  T.
          End:  T.
          EST.DUR:  20.
          UNPLANNED.ANTECEDENTS:  (FLT).
          UNPLANNED.ASSIGN:  (A/C).
     PILOT-DEBRIEF:
          Start:  T.
          End:  T.
          DURATION:  150.
          UNPLANNED.ANTECEDENTS:  (FLT).
          UNPLANNED.ASSIGN:  (PILOT).
     Unplanned tasks:
          A/C-PREP, PILOT-BRIEF, FLT, A/C-SERV,
          PILOT-DEBRIEF.
Resources:
     A/C:
          Name: T.
          Start:  T.
          End:  T.
          UNPLANNED.CONCUR:   A/C-PREP A/C-SERV).
     PILOT:
          Name:  T.
          Start:  T.
          End:  T.
          UNPLANNED.CONCUR:  (PILOT-DEBRIEF PILOT-BRIEF).
     Unplanned resources:
          A/C, PILOT.
```

In the plan shown in Table 2, no values have been entered. The plan is empty. Therefore, no limits have been set, and no task or resource has an EST, LET, LST, or EET property. These will be set by the demons watching the tasks and resources, which are not displayed in Table 2b. These properties are entered only as needed. Also, there are no plannable tasks or resource assignments, and so there is no property PLANNABLE under either TASKS or RESOURCES. The PLANNABLE property is set only when needed by a general task or resource demon.

The model used for the plan given in Table 2 is that shown in Table 1. Note that the estimated durations of A/C-PREP and A/C-SERV, and the durations of PILOT-BRIEF and PILOT-DEBRIEF, have been transferred to the plan, as shown in Table 2b.

As stated, the demons of the model have not been printed. There are many demons set during the creation of the plan illustrated in Table 2. The function PRINT.DEMONLIST can be used to print out selected sets of these demons. It uses two to four arguments. The first two are the name of the process model and the ID of the plan. If only these are present, the demons that are global to the plan are printed. The third argument is TYPE and can be either 'TASKS or 'RESOURCES. If the fourth argument is NIL, it prints the general task or resource demons--i.e., those that act on the tasks or on the resources as a whole. The fourth argument is the name of a task or a resource, depending on the value of TYPE. The function then prints out the local task or resource demons. Use of the function is illustrated in Table 3. The significance of the various terms in the printout is discussed later, when the structure of the demons is discussed.

Table 3
Local Task Demons for FLT in a plan for MISSION

```
(print.demonlist 'mission 1 'tasks 'flt]
The local task demons for FLT are:
     Demon:
          START
          NIL
          PNUMBERP
          LIMIT.FN
          Arg. list:
               (MISSION 1)
               (PILOT-BRIEF A/C-PREP)
               TASKS
               FLT
               TASKS
               LET

     Demon:
          END
          NIL
          PNUMBERP
          LIMIT.FN
          Arg. list:
               (MISSION 1)
               (A/C-SERV PILOT-DEBRIEF)
               TASKS
               FLT
               TASKS
               EST
```

While the details of the demons listed in Table 3 are discussed later, the first demon watches the value of START for the task FLT. If data are entered in this location, then it adds a specification to PILOT-BRIEF and A/C-PREP that has the property name LET, for latest end time, the value of which is the value of the start of FLT. It specifies that these tasks must be completed before the start of FLT. Any violation of this condition will then trigger replanning through another demon.

Similarly, the second demon in Table 3 watches the end of FLT and, when fired, sets an EST, for earliest start time, on the tasks A/C-SERV and PILOT-DEBRIEF.

The functions described in this section are as follows:

```
(PRINT.PLAN
  [LAMBDA (NAME ID STATUS.FLG DEMON.FLG)
    (PROG (PLAN RESOURCES DEMONS)
          (COND
            [(SETQ PLAN (A.GETP.S# PLANS (LIST NAME ID]
            (T (MAPPRINQ ("No plan number " ID
                                                " based on model named "
                                                NAME "." TERPRI))
               (RETURN NIL)))
          (MAPPRINQ ("Plan number " ID " of the model named " NAME
                                          " is as follows:" (RPTQ 2 (TERPRI))
                                          "Tasks:" TERPRI))
          (PRINT.PLAN.1 (A.GETP PLAN (QUOTE TASKS))
                        STATUS.FLG DEMON.FLG)
          (COND
            ((SETQ RESOURCES (A.GETP PLAN (QUOTE RESOURCES)))
              (MAPPRINQ (TERPRI "Resources:" TERPRI))
              (PRINT.PLAN.1 (A.GETP PLAN (QUOTE RESOURCES))
                            STATUS.FLG DEMON.FLG T)))
          (COND
            ([AND DEMON.FLG (SETQ DEMONS (A.GETP PLAN (QUOTE DEMONS]
              (PRINT.DEMONS DEMONS 5)))
          (RETURN (CHARACTER 127])

(PRINT.PLAN.1
  [LAMBDA (LST STATUS.FLG DEMON.FLG RES.FLG)
    (MAPC LST (FUNCTION (LAMBDA (X)
                (PROG (NIL)
                      (COND
                        ((NULL (CDR X))
                          (RETURN NIL))
                        ((EQUAL (CAR X)
                                (QUOTE DEMONS))
                          (COND
                            ((AND DEMON.FLG RES.FLG)
                              (MAPPRINQ ((TAB 5)
                                         "General resource demons:"
                                                                  TERPRI))
                              (PRINT.DEMONS (CDR X)
                                            10))
                            (DEMON.FLG (MAPPRINQ ((TAB 5)
                                                  "General task demons:"
                                                                  TERPRI))
                                       (PRINT.DEMONS (CDR X)
                                                     10)))
                          (RETURN NIL))
((EQUAL (CAR X)
        (QUOTE PLANNABLE))
  (COND
    ((AND STATUS.FLG RES.FLG)
      (MAPPRINQ ((TAB 5)
                 "Plannable resources:  " TERPRI
                 (TAB 10)))
      (MAPRINT (CDR X)
               T NIL (QUOTE %.)
               ", "))
    (STATUS.FLG (MAPPRINQ ((TAB 5)
                           "Plannable tasks:  "
                           TERPRI (TAB 10)))
                (MAPRINT (CDR X)
                         T NIL (QUOTE %.)
                         ", ")))
  (RETURN NIL))
((EQUAL (CAR X)
        (QUOTE UNPLANNED))
  (COND
    ((AND STATUS.FLG RES.FLG)
      (MAPPRINQ ((TAB 5)
                 "Unplanned resources:" TERPRI
                 (TAB 10)))
      (MAPRINT (CDR X)
               T NIL (QUOTE %.)
               ", "))
    (STATUS.FLG (MAPPRINQ ((TAB 5)
                           "Unplanned tasks:"
                           TERPRI (TAB 10)))
                (MAPRINT (CDR X)
                         T NIL (QUOTE %.)
                         ", ")))
  (RETURN NIL))
((EQUAL (CAR X)
        (QUOTE DEFERRED))
  (COND
    ((AND STATUS.FLG RES.FLG)
      (MAPPRINQ ((TAB 5)
                 "Deferred resources:  " TERPRI)))
    (STATUS.FLG (MAPPRINQ ((TAB 5)
                           "Deferred tasks:"
                           TERPRI)))
    (T (RETURN NIL)))
  (TAB 10)
  (MAPRINT (CDR X)
           T NIL (QUOTE %.)
           ", ")
  (RETURN NIL)))
```

```
                    (MAPPRINQ ((TAB 5)
                               (CAR X)
                               ":" TERPRI))
                    (PRINT.PLAN.2 (CDR X)
                                  STATUS.FLG DEMON.FLG RES.FLG])

(PRINT.PLAN.2
  [LAMBDA (LST STATUS.FLG DEMON.FLG RES.FLG)
    (PROG (VAL SUBPLAN)
          (COND
            (RES.FLG (SETQQ SUBPLAN RES.PLAN))
            (T (SETQQ SUBPLAN TASK.PLAN)))
          [COND
            (RES.FLG (MAPPRINQ ((TAB 10)
                                "Name:  "))
                     (COND
                       ([SETQ VAL (A.GETP# LST (QUOTE (RES.PLAN NAME]
                         (MAPPRINQ (VAL "." TERPRI)))
                       (T (PRIN1 "not specified.")
                          (TERPRI]
          (MAPPRINQ ((TAB 10)
                     "Start:  "))
          (COND
            ([ATOM (SETQ VAL (A.GETP# LST (LIST SUBPLAN (QUOTE START]
              (MAPPRINQ ("T." TERPRI)))
            [VAL (MAPPRINQ ((CAR VAL)
                            "." TERPRI (TAB 15)
                            "Priority:  "
                            (A.GETP VAL (QUOTE PRIORITY))
                            TERPRI))
                 (COND
                   ((A.GETP VAL (QUOTE ESTIMATED))
                     (MAPPRINQ ((TAB 15)
                                "Estimated." TERPRI]
            (T (PRIN1 "not specified.")
               (TERPRI)))
          (MAPPRINQ ((TAB 10)
                     "End:  "))
```

```
(COND
  ([ATOM (SETQ VAL (A.GETP# LST (LIST SUBPLAN (QUOTE END]
    (MAPPRINQ ("T." TERPRI)))
  [VAL [MAPPRINQ ((CAR VAL)
                  "." TERPRI (TAB 15)
                  "Priority:  "
                  (A.GETP VAL (QUOTE PRIORITY]
       (COND
         ((A.GETP VAL (QUOTE ESTIMATED))
           (MAPPRINQ ((TAB 15)
                      "Estimated." TERPRI]
  (T (PRIN1 "not specified.")
     (TERPRI)))
(COND
  ((NULL STATUS.FLG)
    (RETURN NIL)))
(MAPC LST (FUNCTION (LAMBDA (X)
         (PROG (NIL)
               (COND
                 ((EQUAL (CAR X)
                         SUBPLAN)
                   (RETURN NIL)))
               [COND
                 ((EQUAL (CAR X)
                         (QUOTE DEMONS))
                   (COND
                     (DEMON.FLG (MAPPRINQ ((TAB 10)
                                           "Local demons:"
                                           TERPRI))
                                (PRINT.DEMONS (CDR X)
                                              15)
                                (RETURN NIL))
                     (T (RETURN NIL]
               (MAPPRINQ ((TAB 10)
                          (CAR X)
                          ":  "))
               (COND
                 ((CDR X)
                   (PRIN1 (CDR X))
                   (PRIN1 "."))
                 (T (PRIN1 "not specified.")))
               (TERPRI])
```

```
(PRINT.DEMONLIST
  [LAMBDA (MODEL.NAME ID TYPE NAME)
    (PROG (PLAN DEMON.LIST TYPE.NAME)
          [COND
            [(SETQ PLAN (A.GETP.S# PLANS (LIST MODEL.NAME ID]
            (T (MAPPRINQ ("No plan with ID " ID " listed under " MODEL
                                              NAME "."))
               (RETURN (CHARACTER 127]
          [COND
            [(NULL TYPE)
              (COND
                ((SETQ DEMON.LIST (A.GETP PLAN (QUOTE DEMONS)))
                  (PRIN1 "The plan demons are:")
                  (TERPRI)
                  (PRINT.DEMONS DEMON.LIST 5))
                (T (PRIN1 "No model demons."]
            [(NULL NAME)
              (COND
                ((EQUAL TYPE (QUOTE TASKS))
                  (SETQQ TYPE.NAME task))
                ((EQUAL TYPE (QUOTE RESOURCES))
                  (SETQQ TYPE.NAME resource)))
              (COND
                ([SETQ DEMON.LIST (A.GETP# PLAN (LIST TYPE
                                                  (QUOTE DEMONS]
                  (MAPPRINQ ("The general " TYPE.NAME " demons are:"
                                             TERPRI))
                  (PRINT.DEMONS DEMON.LIST 5))
                (T (MAPPRINQ ("No general " TYPE.NAME " demons."]
            (T (COND
                 ((EQUAL TYPE (QUOTE TASKS))
                   (SETQQ TYPE.NAME task))
                 ((EQUAL TYPE (QUOTE RESOURCES))
                   (SETQQ TYPE.NAME resource)))
               (COND
                 ([SETQ DEMON.LIST (A.GETP.S# PLAN (LIST TYPE NAME
                                                        (QUOTE DEMONS]
                   (MAPPRINQ ("The local " TYPE.NAME " demons for "
                                            NAME " are:" TERPRI))
                   (PRINT.DEMONS DEMON.LIST 5))
                 (T (MAPPRINQ ("No local demons for " TYPE.NAME NAME
                                                        "."]
          (RETURN (CHARACTER 127]))
```

```
(PRINT.DEMONS
  [LAMBDA (LST N)
    (PROG (M M1)
          (SETQ M (IPLUS N 5))
          (SETQ M1 (IPLUS N 10))
          (MAPC
            LST
            (FUNCTION (LAMBDA (X)
                (PROG (NIL)
                      (TAB N)
                      (MAPPRINQ ("Demon:" TERPRI))
                      (MAPC X (FUNCTION (LAMBDA (Y)
                                 (COND
                                   ((ATOM Y)
                                     (TAB M)
                                     (PRIN1 Y)
                                     (TERPRI))
                                   ((LISTP Y)
                                     (TAB M)
                                     (PRIN1 "Arg. list:  ")
                                     [MAPC Y (FUNCTION (LAMBDA (Z)
                                               (PROG (NIL)
                                                     (TAB M1)
                                                     (PRIN1 Z)
                                                     (TERPRI]
                                     (TERPRI])
```

F. Initial Entry of Data

The function that enters data into a plan is ENTER.VALUE. The plan must exist at the time it is called, otherwise it defaults. It can be used at the top level by the user to enter the original data that define what the required plan is to accomplish. Or it can be used at lower levels to enter data that have been developed by the planning process.

Note that the data are not entered as a simple value, but as (<value> (PRIORITY . <priority>)). If a value of PRIORITY is not given, it is set at zero. The priority that is involved, here, is an internal priority, having nothing to do with a user-defined priority for the mission. Its purpose is to control any necessary replanning, determining which is the preferred direction of replanning. In general, if it is necessary to replan a task that has been completely planned, the preferred replanning will change only the end point that has the highest value of PRIORITY.

ENTER.VALUE also fires the demons. The immediate effect of entering, say, a start value for a given task is to impose a LET on any antecedent task and an LST on any resource assignment with which the task must be concurrent. Similarly, the entry of an end time causes an EST to be placed on any successor task and an EET on any concurrent resource assignment. The values of these limits are the same as that of the entry itself, specifying the same priority.

In addition, if the plan contains a specified duration for a task, demons are present that cause the appropriate end time to be entered when the start time is entered, or they enter the start time when the end time is entered by ENTER.VALUE. The value of PRIORITY for these secondary entries is one higher than that of the original entry. The secondary entries will refire the demons, so that they will cause the entry of the appropriate EST, LST, EET and LET limits. The details of these demons are discussed later.

As an example, Table 4 shows the plan after entering start values for PILOT-BRIEF and A/C-PREP. As before, the model specifies the durations of PILOT-BRIEF and PILOT-DEBRIEF, and the estimated durations of A/C-PREP and A/C-SERV. Therefore, the entry of start times causes the entry of the corresponding end times, and the corresponding EST for FLT.

The entry of times for A/C-PREP and PILOT-BRIEF causes the entry of LST and EET values for the resource assignments, A/C and PILOT. It can be observed that the EET values are inappropriate since there are no times yet given to A/C-SERV and PILOT-DEBRIEF. This is of little consequence, however, since the assignment of these resources will not be sought as long as they have non-null values of UNPLANNED.CONCUR.

Note also that the end time of A/C-PREP includes the notation that it is estimated. The actual form of this component is (END 210 (PRIORITY . 1) (ESTIMATED . T)), which has been recognized by the print function. This notation has been carried along into the EST of FLT (since the estimated end time of A/C-PREP is after the actual end time of PILOT-BRIEF) and the EET of the A/C assignment.

One further complication needs to be noted, although detailed discussion of it is deferred until later. Suppose we sought to enter the end of PILOT-BRIEF as, say, 90. Its duration is given as 120, so the indicated start would be -30. If the current time used by the system, given by S.CLOCK, which is a global variable of the system, is 0 (as it is assumed to be throughout this report), this would set the start before current time. The demons would recognize this fact and the start time would actually be entered as 0 with priority -1. This would lead to the end time being shifted to 120, also with priority -1. In general, a priority of -1 associated with a value indicates that the value is forced by the constraints of the model and the current time of the system.

In the situation discussed above, where an attempt is made to enter a value that is inconsistent with current time, a separate demon is fired which prints an alert message. This demon is set by ENTER.VALUE to guard the values that are entered by the user, and is fired whenever those values are changed as a result of replanning or other operation. This demon is also discussed later.

Table 4
Plan for MISSION after Entry of Start Times
for A/C-PREP and PILOT-BRIEF.
(Local Demons Only)

```
(enter.value 'mission 1 'tasks 'a/c-prep 'start 120]
OK
(enter.value 'mission 1 'tasks 'pilot-brief 'start 60]
OK
(print.plan 'mission 1 t]
Plan number 1 of the model named MISSION is as follows:

Tasks:
     A/C-PREP:
          Start:  120.
               Priority:  0
          End:  210.
               Priority:  1
               Estimated.
          EST.DUR:  90.
          UNPLANNED.SUCCESSORS:  (FLT).
          UNPLANNED.ASSIGN:  (A/C).
     PILOT-BRIEF:
          Start:  60.
               Priority:  0
          End:  180.
               Priority:  1
          DURATION:  120.
          UNPLANNED.SUCCESSORS:  (FLT).
          UNPLANNED.ASSIGN:  (PILOT).
     FLT:
          Start:  T.
          End:  T.
          UNPLANNED.SUCCESSORS:  (A/C-SERV PILOT-DEBRIEF).
          EST:  (210 (PRIORITY . 1)).
     A/C-SERV:
          Start:  T.
          End:  T.
          EST.DUR:  20.
          UNPLANNED.ANTECEDENTS:  (FLT).
          UNPLANNED.ASSIGN:  (A/C).
     PILOT-DEBRIEF:
          Start:  T.
          End:  T.
          DURATION:  150.
          UNPLANNED.ANTECEDENTS:  (FLT).
          UNPLANNED.ASSIGN:  (PILOT).
```

Table 4
(Continued)

```
          Unplanned tasks:
               A/C-PREP, PILOT-BRIEF, FLT, A/C-SERV,
               PILOT-DEBRIEF.
          Plannable tasks:
               FLT.
          Deferred tasks:
               A/C-PREP.
     Resources:
          A/C:
               Name:  T.
               Start:  T.
               End:  T.
               UNPLANNED.CONCUR:   (A/C-SERV).
               LST:  (120 (PRIORITY . 0)).
               EET:  (210 (PRIORITY . 1) (ESTIMATED . T)).
          PILOT:
               Name:  T.
               Start:  T.
               End:  T.
               UNPLANNED.CONCUR:   (PILOT-DEBRIEF).
               LST:  (60 (PRIORITY . 0)).
               EET:  (180 (PRIORITY . 1)).
          Unplanned resources:
               A/C, PILOT.
          Plannable resources:
               A/C, PILOT.
```

Note that PILOT-BRIEF has been removed from the list of unplanned tasks. It is now completely planned. A/C-PREP has not been removed since its end value is only estimated. However, it has been added as a deferred task--its planning can be left until later.

Note also that both tasks have been deleted from the list of unplanned antecedents of FLT. In fact, there are no unplanned antecedents and the list is removed. Taken literally, this is not true, since A/C-PREP has not yet been planned. However, the purpose of recording the unplanned antecedents and successors is to guide the selection of the next task to be planned. For this purpose, A/C-PREP should be treated as if it had been planned; this is the reason for using estimated durations.

The actual entry of the data is made by P.ENTRY which also fires the demons in sequence: local demons, task and resource demons, and plan demons. The structure of the demons, the way they are fired, and the further consequences of their being fired, are discussed in subsequent sections. Note, however, that we define three classes of demons. Local demons are attached to individual tasks or resource assignments and watch the data held in the TASK.PLAN or RES.PLAN for the task or assignment. The type demons are attached to the TASKS and RESOURCES parts of the plan and are used to maintain the mutual consistency of the data in the separate tasks or assignments. The global demons are attached to the plan as a whole and are used to drive the planning process to completion, or as far as possible.

The function ENTER.DURATION is included here because it may cause the entry of data. If, for example, a start time of FLT has been entered, it has the effect of causing the end time to be entered. It is not equivalent to the direct entry of an end time via ENTER.VALUE, since it also sets a demon that will maintain the duration during any subsequent replanning. Also, it does not set the demon that prints an alert message if inputted values are later changed.

ENTER.DURATION can also be used prior to the entry of any value. In this case, it acts as a modification of the model for the given plan. That is, it has the same effect as if the plan specified the given duration for the given task.

It can also be used after the entry of both the start and end times. If these values are not consistent with the duration being entered, it forces replanning as is discussed later.

Note that ENTER.DURATION overrides any conflicting specification by the model, or, for that matter, a previous specification through ENTER.DURATION. It gives the user complete command of this requirement.

Two other functions may be mentioned at this time. P.ENTRY.1 takes a subplan that is either of the form

```
(TASKS (START . <time>) (END . <time>))
```

or

```
(RESOURCES (NAME . <name>)(START . <time>)(END . <time>)).
```

P.ENTRY.1 is intended for use when a task or resource assignment has been planned or assigned by another planner or a scheduler. The forms given above for SUBPLAN are those returned in response to a request for a plan or assignment.

P.ENTRY.2 is a truncated form of P.ENTRY. It assumes that the entry is into a task. It fires only the local demons. It is used by some of the demons, as is discussed later.

The demons are fired by P.ENTRY calling FIRE.P.DEMONS which, in turn, calls FIRE.LOCAL.DEMONS, FIRE.TYPE.DEMONS, and FIRE.GL.DEMONS (for fire global demons). The structure of these demons is discussed later, and these functions are therefore listed here without further comment. What is important to note, however, is the order of their firing-- first the local demons, then the type demons, and finally the demons that are global for the plan.

The functions discussed here are defined as follows:

```
(ENTER.VALUE
  [LAMBDA (MODEL.NAME ID TYPE NAME PROP VAL PRIORITY PRINT.SUP.FLG)
    (PROG (MODEL PLAN)
          (COND
            ((NOT (NUMBERP VAL))
              (MAPPRINQ
                ("Incorrectly entered.  Value not a number.  Re-enter."
                           TERPRI))
              (RETURN NIL)))
          (COND
            ((SETQ MODEL (A.GETP MODELS MODEL.NAME)))
            (T (MAPPRINQ ("No model named " MODEL.NAME "." TERPRI))
              (RETURN NIL)))
          (COND
            [(SETQ PLAN (A.GETP.S# PLANS (LIST MODEL.NAME ID]
            (T (MAPPRINQ ("No plan with ID " ID " under " MODEL.NAME
                          "." TERPRI)
               (RETURN NIL)))
          (COND
            [(OR (EQUAL TYPE (QUOTE TASKS))
                 (EQUAL TYPE (QUOTE RESOURCES]
            (T (MAPPRINQ ("Do not understand " TYPE
                                     ".  Must be either TASKS or RESOURCES."
                                                  TERPRI))
               (RETURN NIL)))
          (COND
            ((NULL PRIORITY)
              (SETQ PRIORITY 0)))
          (SETQ VAL (LIST VAL (CONS (QUOTE PRIORITY)
                                     PRIORITY)))
          (P.ENTRY PLAN TYPE NAME PROP VAL (NULL PRINT.SUP.FLG)
                   MODEL.NAME ID)
          (COND
            (PRINT.SUP.FLG (RETURN T))
            (T (PRINT1 "OK")
               (RETURN (CHARACTER 127])

(ENTER.DURATION
  [LAMBDA (MODEL.NAME ID TYPE NAME DURATION PRINT.SUP.FLG)
    (PROG (MODEL PLAN TYPE.SPECS TYPE.PLAN START.VAL END.VAL ARG.LIST
                 SUBLIST)
          (COND
            ((NOT (NUMBERP DURATION))
              (MAPPRINQ
              ("Incorrectly entered.  Duration not a number.  Re-enter."
                         TERPRI))
              (RETURN NIL)))
          (COND
            ((SETQ MODEL (A.GETP MODELS MODEL.NAME)))
            (T (MAPPRINQ ("No model named " MODEL.NAME "." TERPRI))
               (RETURN NIL)))
          (COND
            [(SETQ PLAN (A.GETP.S# PLANS (LIST MODEL.NAME ID]
            (T (MAPPRINQ ("No plan with ID " ID " for model "
                                                MODEL.NAME "." TERPRI))
               (RETURN NIL)))
          (COND
            ((EQUAL TYPE (QUOTE TASKS))
              (SETQQ TYPE.SPECS TASK.SPECS)
              (SETQQ TYPE.PLAN TASK.PLAN))
            ((EQUAL TYPE (QUOTE RESOURCES))
              (SETQQ TYPE.SPECS RES.SPECS)
              (SETQQ TYPE.PLAN RES.PLAN))
            (T (MAPPRINQ (
"Do not understand.  3rd argument must be TASKS or RESOURCES." TERPRI))
               (RETURN NIL)))
          (KILL.DURATION.DEMONS PLAN TYPE NAME)
          [SETQ START.VAL (A.GETP# PLAN (LIST TYPE NAME TYPE.PLAN
                                              (QUOTE START]
          [SETQ END.VAL (A.GETP# PLAN (LIST TYPE NAME TYPE.PLAN
                                            (QUOTE END]
          (A.PUT# PLAN (LIST TYPE NAME (QUOTE DURATION))
                  DURATION)
```

```
[COND
  ((OR (NULL (LISTP START.VAL))
       (NULL (LISTP END.VAL)))
    (SETQ ARG.LIST (LIST (LIST MODEL.NAME ID TYPE NAME
                               (QUOTE START)
                               NIL
                               (QUOTE PNUMBERP)
                               (QUOTE SET.END))
                         (LIST MODEL.NAME ID TYPE NAME
                               (QUOTE END)
                               NIL
                               (QUOTE PNUMBERP)
                               (QUOTE SET.START))
                         NAME TYPE (QUOTE DURATION)
                         DURATION))
    (SETQ SUBLIST (LIST NAME TYPE (QUOTE DURATION)
                        DURATION)))
  (T [SETQ ARG.LIST (LIST (LIST MODEL.NAME ID TYPE NAME T NIL
                                (QUOTE DUMMY)
                                (QUOTE DUR.WATCH]
     (SETQ SUBLIST (LIST NAME TYPE (QUOTE DURATION)
                         DURATION]
(COND
  ((AND (NULL (LISTP START.VAL))
        (NULL (LISTP END.VAL)))
    (SET.LOCAL.DEMON PLAN TYPE NAME (QUOTE START)
                     NIL
                     (QUOTE PNUMBERP)
                     (QUOTE SET.END)
                     ARG.LIST)
    (SET.LOCAL.DEMON PLAN TYPE NAME (QUOTE END)
                     NIL
                     (QUOTE PNUMBERP)
                     (QUOTE SET.START)
                     ARG.LIST))
  ((NULL (LISTP START.VAL))
    (SET.START NIL (A.GETP# PLAN (LIST TYPE NAME TYPE.PLAN))
               ARG.LIST
               (QUOTE END)
               SUBLIST))
  ((NULL (LISTP END.VAL))
    (SET.END NIL (A.GETP# PLAN (LIST TYPE NAME TYPE.PLAN))
             ARG.LIST
             (QUOTE START)
             SUBLIST))
  (T (DUR.WATCH (A.GETP# PLAN (LIST TYPE NAME TYPE.PLAN))
                NIL ARG.LIST SUBLIST)))
```

```
          (FIRE.TYPE.DEMONS (A.GETP# PLAN (LIST TYPE (QUOTE DEMONS))
                                          TYPE NAME))
          (FIRE.GL.DEMONS (A.GETP PLAN (QUOTE DEMONS))
                          TYPE)
          (COND
            (PRINT.SUP.FLG (RETURN NIL))
            (T (PRINT1 "OK")
               (RETURN (CHARACTER 127])

(P.ENTRY
  [LAMBDA (PLAN TYPE NAME PROP VAL WATCH.FLG MODEL.NAME ID)
    (PROG (PLAN.TYPE OLD.VAL NEW.VAL)
          (COND
            ((EQUAL TYPE (QUOTE TASKS))
              (SETQQ PLAN.TYPE TASK.PLAN))
            ((EQUAL TYPE (QUOTE RESOURCES))
              (SETQQ PLAN.TYPE RES.PLAN)))
          (SETQ NEW.VAL (A.GETP# PLAN (LIST TYPE NAME PLAN.TYPE)))
          (SETQ OLD.VAL (COPY NEW.VAL))
          (A.PUT NEW.VAL PROP VAL)
          [COND
            (WATCH.FLG (SET.LOCAL.DEMON PLAN TYPE NAME PROP (CAR VAL)
                                        (QUOTE CHANGE)
                                        (QUOTE PRINT.CHANGE.ALERT)
                                        (LIST (LIST MODEL.NAME ID)
                                              TYPE NAME PROP (CAR VAL]
          (FIRE.P.DEMONS PLAN TYPE NAME NEW.VAL OLD.VAL PROP])

(P.ENTRY.1
  [LAMBDA (PLAN NAME.LIST SUBPLAN)
    (PROG (TYPE PLAN.TYPE NAME NEW.VAL OLD.VAL DEMONLIST)
          (COND
            ((EQUAL (SETQ TYPE (CAR NAME.LIST))
                    (QUOTE TASKS))
              (SETQQ PLAN.TYPE TASK.PLAN))
            (T (SETQQ PLAN.TYPE RES.PLAN)))
          (SETQ NAME (2ND NAME.LIST))
          (SETQ NEW.VAL (A.GETP# PLAN (LIST TYPE NAME PLAN.TYPE)))
          (SETQ OLD.VAL (COPY NEW.VAL))
          (A.PUT NEW.VAL (QUOTE START)
                 (A.GETP SUBPLAN (QUOTE START)))
          [COND
            ([SETQ DEMONLIST (A.GETP# PLAN (LIST TYPE NAME
                                                 (QUOTE DEMONS]
              (FIRE.LOCAL.DEMONS DEMONLIST NEW.VAL OLD.VAL (QUOTE START]
            (A.PUT NEW.VAL (QUOTE END)
                   (A.GETP SUBPLAN (QUOTE END)))
            [COND
              (DEMONLIST (FIRE.LOCAL.DEMONS DEMONLIST NEW.VAL OLD.VAL
                                            (QUOTE END]
            [COND
              ((EQUAL TYPE (QUOTE RESOURCES))
                [A.PUT NEW.VAL (QUOTE NAME)
                       (CAR (A.GETP SUBPLAN (QUOTE NAME]
                (COND
                  (DEMONLIST (FIRE.LOCAL.DEMONS DEMONLIST NEW.VAL OLD.VAL
                                                (QUOTE NAME]
            (COND
              ([SETQ DEMONLIST (A.GETP# PLAN (LIST TYPE (QUOTE DEMONS]
                (FIRE.TYPE.DEMONS DEMONLIST TYPE NAME)))
            (COND
              ((SETQ DEMONLIST (A.GETP PLAN (QUOTE DEMONS)))
                (FIRE.GL.DEMONS DEMONLIST TYPE])

(P.ENTRY.2
  [LAMBDA (T.PLAN TASK PROP VAL.LIST)
    (PROG (OLD.VAL NEW.VAL DEMONLIST)
            [SETQ NEW.VAL (A.GETP# T.PLAN (LIST TASK (QUOTE TASK.PLAN]
            (SETQ OLD.VAL (COPY NEW.VAL))
            (A.PUT NEW.VAL PROP VAL.LIST)
            (COND
              ([SETQ DEMONLIST (A.GETP# T.PLAN (LIST TASK (QUOTE DEMONS]
                (FIRE.LOCAL.DEMONS DEMONLIST NEW.VAL OLD.VAL PROP])

(FIRE.P.DEMONS
  [LAMBDA (PLAN TYPE NAME NEW.VAL OLD.VAL PROP)
    (PROG (DEMONLIST)
            (COND
              ([AND NAME (SETQ DEMONLIST (A.GETP# PLAN (LIST TYPE NAME
                                                      (QUOTE DEMONS]
                (FIRE.LOCAL.DEMONS DEMONLIST NEW.VAL OLD.VAL PROP)))
            (COND
              ([AND TYPE (SETQ DEMONLIST (A.GETP# PLAN (LIST TYPE
                                                      (QUOTE DEMONS]
                (FIRE.TYPE.DEMONS DEMONLIST TYPE NAME)))
            (COND
              ((SETQ DEMONLIST (A.GETP PLAN (QUOTE DEMONS)))
                (FIRE.GL.DEMONS DEMONLIST TYPE])
```

```
(FIRE.LOCAL.DEMONS
  [LAMBDA (DEMON.LIST NEW.VAL OLD.VAL PROP)
    (MAPC DEMON.LIST (FUNCTION (LAMBDA (X)
              (COND
                ((OR (EQUAL (CAR X)
                            T)
                     (AND (EQUAL PROP (CAR X))
                          (APPLY* (3RD X)
                                  (CAR X)
                                  (2ND X)
                                  NEW.VAL)))
                  (APPLY* (4TH X)
                          OLD.VAL NEW.VAL (5TH X)
                          PROP])

(FIRE.TYPE.DEMONS
  [LAMBDA (DEMONLIST TYPE NAME)
    (MAPC DEMONLIST (FUNCTION (LAMBDA (X)
              (COND
                ((APPLY* (2ND X)
                         (CAR X)
                         TYPE NAME)
                  (APPLY* (3RD X)
                          (CAAR (4TH X))
                          (2ND (CAR (4TH X)))
                          TYPE NAME])

(FIRE.GL.DEMONS
  [LAMBDA (DEMONLIST TYPE)
    (MAPC DEMONLIST (FUNCTION (LAMBDA (X)
              (COND
                ((APPLY* (2ND X)
                         (CAR X)
                         TYPE)
                  (APPLY* (3RD X)
                          (4TH X)
                          TYPE])
```

G. Local Sequential and Concurrency Demons

Demons are set when the plan is first created by MAKE.PLAN on its call of MAKE.PLAN.DEMONS.

MAKE.PLAN.DEMONS calls SET.SEQ.DEMONS and SET.CONCUR.DEMONS. The former sets the demons that enforce the sequential constraints among the tasks, the latter the concurrency relations between assignments and tasks. MAKE.PLAN.DEMONS also calls SET.RES.DEMONS, SET.DURATION.DEMONS, SET.TASK.DEMONS, and SET.P.DEMONS, which set other demons whose effects are discussed later.

SET.SEQ.DEMONS and SET.CONCUR.DEMONS both use SET.CONSTRAINT.-DEMON to set up the required demon. It is actually entered into the plan by SET.LOCAL.DEMON which is a general purpose function for entering a local demon after the argument list has been constructed.

The structure of a local demon is defined as it was for the schedulers (cf Technical Report 14) as a list of five elements:

```
(<property> <value> <function.1> <function.2> <arg. list>)
```

The first three terms define the precondition of the demon, the last two the function it calls--and its arguments--when fired.

The local demons discussed here differ from those defined for the schedulers in two ways. The precondition is handled slightly differently, and the argument list is different.

The precondition differs in the way it is tested in the function FIRE.LOCAL.DEMONS, called, for example, by ENTER.VALUE. The precondition has two parts. The first part requires a match between the PROP of ENTER.VALUE and the first element in the demon's argument list. The form of the second part is the same as that used in the scheduler demons. It requires that

```
(<function.1> (CAR <arg. list>) (2ND <arg. list>) VAL)
```

be non-NIL, where VAL is the value being entered by ENTER.VAL. In the demons currently being used, the second element of the argument list is NIL. The position is maintained both for similarity with the scheduler's demons, and in case it should prove useful later.

The demons discussed here have no need for the self or mutual destruct properties that dominated the structure of the argument list of the demons used in the schedulers. These demons do not change the data to which they are attached, and so will not refire themselves. Further, there seems to be no reason why they should ever be deleted from a plan, and, if it should become necessary to do so, each is attached to a single unique data element. Therefore, we have simplified the argument list that is given to them. It is specified to have the following form:

```
(<list> <target list> <target type> <source> <source type>
        <limit property>)
```

The first term is a list of auxiliary information, specifically the model name and the ID of the plan, so that the plan can be recovered from PLANS. The second element is a list of the tasks or resources that are to be affected by the demon. The third element is either TASKS or RESOURCES, depending on whether the target list is a list of tasks or resource assignments. The fourth element is the task or resource to which the demon is attached, and the fifth either TASKS or RESOURCES depending on which the source is. The final element is EST, LST, EET, or LET, depending on what limiting condition is to be imposed on the elements of the target list.

These demons are illustrated in Table 2, given previously.

The preconditions of these demons use the function PNUMBERP. PNUMBERP returns (NUMBERP (CAR VAL)) since VAL is not a simple number. VAL, in fact, has either the form (<number> (PRIORITY . <number>)), or (<number> (PRIORITY . <number>) (ESTIMATED . T)), as discussed later.

The function used by these local demons is LIMIT.FN. Its complications are principally caused by the fact that it is used under a relatively wide variety of conditions, the demon being fired by the addition of a start or end time to a task or resource, and entering limits on a set of either tasks or resources.

LIMIT.FN also does some of the other housekeeping required. If SOURCE is the task or assignment that is causing the limit condition to be set, and TARGET a task or assignment on which it is being set, then SOURCE is removed as an unplanned antecedent, successor, assign, or concur, as appropriate, from each TARGET. Also, the TARGET is now listed as plannable, if it was not before, providing it is not already completely planned.

In setting a limit, ENTER.LIMIT is called. This function determines whether there is already a value set for that limit. If not, the limit is entered directly. If a value already exists, and if the new value is a tighter constraint than the old one, the new value is used. If it is looser, it calls ENTER.TASK.LIMIT or ENTER.RES.LIMIT, as appropriate, to check if the old limit is still valid.

The need to check the old limit arises from replanning considerations. A given limit may have been originally set on, say, the start of FLT by the end time of PILOT-BRIEF. If PILOT-BRIEF is replanned, its end value may be changed in a way that permits a looser constraint on the start of FLT. Hence, in this case, ENTER.TASK.LIMIT is called. It examines the end times of all the antecedents of FLT and determines what limit should actually be placed on the start of FLT. If a looser constraint can be entered, it makes the entry.

These functions are as follows:

```
(MAKE.PLAN.DEMONS
  [LAMBDA (MODEL.NAME ID)
    (PROG (MODEL PLAN RESOURCES)
          (COND
            ((SETQ MODEL (A.GETP MODELS MODEL.NAME)))
            (T (MAPPRINQ ("No model named " MODEL.NAME "." TERPRI))
               (RETURN NIL)))
          (COND
            [(SETQ PLAN (A.GETP.S# PLANS (LIST MODEL.NAME ID]
            (T (MAPPRINQ ("No plan with ID " ID " for model named "
                                                MODEL.NAME "." TERPRI))
               (RETURN NIL)))
          (SET.SEQ.DEMONS MODEL.NAME ID PLAN (A.GETP MODEL
                                                         (QUOTE TASKS))
                          (A.GETP MODEL (QUOTE TASK.SPECS)))
          (COND
            ((SETQ RESOURCES (A.GETP MODEL (QUOTE RESOURCES)))
             (SET.CONCUR.DEMONS MODEL.NAME ID PLAN RESOURCES
                                (A.GETP MODEL (QUOTE RES.SPECS)))
             (SET.RES.DEMONS MODEL.NAME ID PLAN)))
          (SET.DURATION.DEMONS MODEL.NAME ID MODEL PLAN RESOURCES)
          (SET.TASK.DEMONS MODEL.NAME ID PLAN)
          (SET.P.DEMONS MODEL.NAME ID PLAN])
```

```
(SET.SEQ.DEMONS
  [LAMBDA (MODEL.NAME ID PLAN TASK.LIST TASK.SPECS)
    (MAPC TASK.LIST (FUNCTION (LAMBDA (X)
              (PROG (ANTECEDENTS SUCCESSORS ASSIGN)
                    [COND
                      ([SETQ ANTECEDENTS (A.GETP# TASK.SPECS
                                                  (LIST X
                                                    (QUOTE ANTECEDENTS]
                        (SET.CONSTRAINT.DEMON PLAN (QUOTE TASKS)
                                              X
                                              (QUOTE START)
                                              (QUOTE LIMIT.FN)
                                              MODEL.NAME ID ANTECEDENTS
                                              (QUOTE TASKS)
                                              (QUOTE LET]
                    [COND
                      ([SETQ SUCCESSORS (A.GETP# TASK.SPECS
                                                 (LIST X
                                                   (QUOTE SUCCESSORS]
                        (SET.CONSTRAINT.DEMON PLAN (QUOTE TASKS)
                                              X
                                              (QUOTE END)
                                              (QUOTE LIMIT.FN)
                                              MODEL.NAME ID SUCCESSORS
                                              (QUOTE TASKS)
                                              (QUOTE EST]
                    (COND
                      ([SETQ ASSIGN (A.GETP# TASK.SPECS (LIST X
                                                         (QUOTE ASSIGN]
                        (SET.CONSTRAINT.DEMON PLAN (QUOTE TASKS)
                                              X
                                              (QUOTE START)
                                              (QUOTE LIMIT.FN)
                                              MODEL.NAME ID ASSIGN
                                              (QUOTE RESOURCES)
                                              (QUOTE LST))
                        (SET.CONSTRAINT.DEMON PLAN (QUOTE TASKS)
                                              X
                                              (QUOTE END)
                                              (QUOTE LIMIT.FN)
                                              MODEL.NAME ID ASSIGN
                                              (QUOTE RESOURCES)
                                              (QUOTE EET])
```

```
(SET.CONCUR.DEMONS
  [LAMBDA (MODEL.NAME ID PLAN RESOURCES RES.SPECS)
    (MAPC RESOURCES (FUNCTION (LAMBDA (X)
               (PROG (CONCURRENT)
                     (COND
                       ([SETQ CONCURRENT (A.GETP# RES.SPECS (LIST X
                                                     (QUOTE CONCURRENT]
                         (SET.CONSTRAINT.DEMON PLAN (QUOTE RESOURCES)
                                               X
                                               (QUOTE START)
                                               (QUOTE LIMIT.FN)
                                               MODEL.NAME ID CONCURRENT
                                               (QUOTE TASKS)
                                               (QUOTE EST))
                         (SET.CONSTRAINT.DEMON PLAN (QUOTE RESOURCES)
                                               X
                                               (QUOTE END)
                                               (QUOTE LIMIT.FN)
                                               MODEL.NAME ID CONCURRENT
                                               (QUOTE TASKS)
                                               (QUOTE LET])

(SET.CONSTRAINT.DEMON
  [LAMBDA (PLAN TYPE NAME PROP FN MODEL.NAME ID LST TARGET.TYPE
                LIMIT.TYPE)
    (SET.LOCAL.DEMON PLAN TYPE NAME PROP NIL (QUOTE PNUMBERP)
                     FN
                     (LIST (LIST MODEL.NAME ID)
                           LST TARGET.TYPE NAME TYPE LIMIT.TYPE])

(SET.LOCAL.DEMON
  [LAMBDA (PLAN TYPE NAME PROP VAL FN1 FN2 ARG.LIST)
    (A.ADDPROP# PLAN (LIST TYPE NAME (QUOTE DEMONS))
                (LIST PROP VAL FN1 FN2 ARG.LIST])
```

```
(LIMIT.FN
  [LAMBDA (OLD.VAL NEW.VAL ARG.LIST PROP)
    (PROG (SOURCE TARGET.TYPE LIMIT.TYPE PLAN PLAN.TYPE FLG SOURCE.TYPE
                UNPLANNED.TYPE PLANNABLE)
          (SETQ PLAN (A.GETP.S# PLANS (CAR ARG.LIST)))
          (SETQ SOURCE (4TH ARG.LIST))
          (SETQ TARGET.TYPE (3RD ARG.LIST))
          (COND
            ((OR (EQUAL (SETQ LIMIT.TYPE (6TH ARG.LIST))
                        (QUOTE EST))
                 (EQUAL LIMIT.TYPE (QUOTE EET)))
              (SETQ FLG T)))
          (COND
            ((EQUAL (SETQ SOURCE.TYPE (5TH ARG.LIST))
                    (QUOTE RESOURCES))
              (SETQQ UNPLANNED.TYPE UNPLANNED.ASSIGN))
            ((EQUAL TARGET.TYPE (QUOTE RESOURCES))
              (SETQQ UNPLANNED.TYPE UNPLANNED.CONCUR))
            ((EQUAL PROP (QUOTE END))
              (SETQQ UNPLANNED.TYPE UNPLANNED.ANTECEDENTS))
            (T (SETQQ UNPLANNED.TYPE UNPLANNED.SUCCESSORS)))
          (COND
            ((EQUAL TARGET.TYPE (QUOTE TASKS))
              [SETQ PLANNABLE (A.GETP# PLAN (QUOTE (TASKS PLANNABLE]
              (SETQQ PLAN.TYPE TASK.PLAN))
            ((EQUAL TARGET.TYPE (QUOTE RESOURCES))
              [SETQ PLANNABLE (A.GETP# PLAN (QUOTE (RESOURCES PLANNABLE]
              (SETQQ PLAN.TYPE RES.PLAN)))
          (MAPC (2ND ARG.LIST)
                (FUNCTION (LAMBDA (X)
                    (PROG (UNPLANNED SUBPLAN)
                          (ENTER.LIMIT PLAN TARGET.TYPE X
                                       (A.GETP NEW.VAL PROP)
                                       LIMIT.TYPE FLG (CAR ARG.LIST))
                          (SETQ SUBPLAN (A.GETP# PLAN (LIST TARGET.TYPE
                                                              X
                                                              PLAN.TYPE)))
                          (COND
                            ((EQUAL (SETQ UNPLANNED
                                      (A.GETP# PLAN (LIST TARGET.TYPE X
                                                          UNPLANNED.TYPE)))
                                    (LIST SOURCE))
                              (A.REMPROP# PLAN (LIST TARGET.TYPE X
                                                     UNPLANNED.TYPE)))
                            (T (A.REMVAL# PLAN (LIST TARGET.TYPE X
                                                     UNPLANNED.TYPE)
                                          SOURCE)))
                          (COND
                            [(AND (EQUAL TARGET.TYPE (QUOTE TASKS))
                                  (LISTP (A.GETP SUBPLAN (QUOTE START)))
                                  (LISTP (A.GETP SUBPLAN (QUOTE END]
                            [(AND (EQUAL TARGET.TYPE (QUOTE RESOURCES))
                                  (NULL (EQ (A.GETP SUBPLAN
                                                    (QUOTE NAME))
                                            T))
                                  (LISTP (A.GETP SUBPLAN (QUOTE START)))
                                  (LISTP (A.GETP SUBPLAN (QUOTE END]
                            ((OR (NULL PLANNABLE)
                                 (NOT (MEMBER X PLANNABLE)))
                              (COND
                                ((EQUAL TARGET.TYPE (QUOTE TASKS))
                                  (A.ADDPROP# PLAN (QUOTE (TASKS
                                                          PLANNABLE))
                                              X))
                                (T (A.ADDPROP# PLAN (QUOTE (RESOURCES
                                                          PLANNABLE))
                                               X])

(ENTER.LIMIT
  [LAMBDA (PLAN TYPE NAME NEW.VAL LIMIT FLG ARG.LIST)
    (PROG (OLD.VAL)
          (COND
            ([NULL (SETQ OLD.VAL (A.GETP# PLAN (LIST TYPE NAME LIMIT]
              (A.PUT# PLAN (LIST TYPE NAME LIMIT)
                      NEW.VAL))
            ((EQUAL (CAR NEW.VAL)
                    (CAR OLD.VAL)))
            ((EQ TYPE (QUOTE TASKS))
              (ENTER.TASK.LIMIT PLAN NAME LIMIT NEW.VAL FLG ARG.LIST))
            ((EQ TYPE (QUOTE RESOURCES))
              (ENTER.RES.LIMIT PLAN NAME LIMIT NEW.VAL FLG ARG.LIST])

(ENTER.TASK.LIMIT
  [LAMBDA (PLAN NAME LIMIT NEW.VAL FLG ARG.LIST)
    (PROG (MODEL TRIAL FLG1 TRIAL.LIST)
          (SETQ MODEL (A.GETP MODELS (CAR ARG.LIST)))
          (SETQ TRIAL (COPY NEW.VAL))
```

```
(COND
  ((OR (EQ LIMIT (QUOTE EST))
       (EQ LIMIT (QUOTE LST)))
    [SETQ TRIAL.LIST (A.GETP# MODEL (LIST (QUOTE TASK.SPECS)
                                          NAME
                                          (QUOTE ANTECEDENTS]
    (SETQ FLG1 NIL))
  (T [SETQ TRIAL.LIST (A.GETP# MODEL (LIST (QUOTE TASK.SPECS)
                                           NAME
                                           (QUOTE SUCCESSORS]
     (SETQ FLG1 T)))
[COND
  (TRIAL.LIST
    (MAPC TRIAL.LIST
          (FUNCTION (LAMBDA (X)
              (PROG (NEW.TRIAL)
                    [COND
                      [FLG1 (SETQ NEW.TRIAL
                              (A.GETP# PLAN (LIST
                                                (QUOTE TASKS)
                                                X
                                               (QUOTE TASK.PLAN)
                                                (QUOTE START]
                      (T (SETQ NEW.TRIAL (A.GETP#
                               PLAN
                               (LIST (QUOTE TASKS)
                                     X
                                     (QUOTE TASK.PLAN)
                                     (QUOTE END]
                    (COND
                      ((NULL (LISTP NEW.TRIAL)))
                      ([OR (AND FLG (GREATERP (CAR NEW.TRIAL)
                                              (CAR TRIAL)))
                           (AND (NULL FLG)
                                (LESSP (CAR NEW.TRIAL)
                                       (CAR TRIAL]
                        (SETQ TRIAL (COPY NEW.TRIAL]
(COND
  ([OR (EQ (CAR TRIAL)
           (CAR NEW.VAL))
       (AND FLG (GREATERP (CAR TRIAL)
                          (CAR NEW.VAL)))
       (AND (NULL FLG)
            (LESSP (CAR TRIAL)
                   (CAR NEW.VAL]
    (A.PUT# PLAN (LIST (QUOTE TASKS)
                       NAME LIMIT)
            (CONS (CAR TRIAL)
                  (CDR NEW.VAL])
```

```
(ENTER.RES.LIMIT
  [LAMBDA (PLAN NAME LIMIT NEW.VAL FLG ARG.LIST)
    (PROG (MODEL TRIAL FLG1 TRIAL.LIST)
          (SETQ MODEL (A.GETP MODELS (CAR ARG.LIST)))
          (SETQ TRIAL (COPY NEW.VAL))
          (COND
            ((OR (EQ LIMIT (QUOTE EET))
                 (EQ LIMIT (QUOTE LET)))
              (SETQ FLG1 T)))
          [COND
            ([SETQ TRIAL.LIST (A.GETP# MODEL (LIST (QUOTE RES.SPECS)
                                                   NAME
                                                   (QUOTE CONCURRENT]
              (MAPC TRIAL.LIST
                    (FUNCTION (LAMBDA (X)
                        (PROG (NEW.TRIAL)
                              [COND
                                [FLG1 (SETQ NEW.TRIAL
                                        (A.GETP# PLAN (LIST
                                                        (QUOTE TASKS)
                                                        X
                                                       (QUOTE TASK.PLAN)
                                                        (QUOTE END]
                                (T (SETQ NEW.TRIAL (A.GETP#
                                       PLAN
                                       (LIST (QUOTE TASKS)
                                             X
                                             (QUOTE TASK.PLAN)
                                             (QUOTE START]
                              (COND
                                ((NULL (LISTP NEW.TRIAL)))
                                ([OR (AND FLG (GREATERP (CAR NEW.TRIAL)
                                                        (CAR TRIAL)))
                                     (AND (NULL FLG)
                                          (LESSP (CAR NEW.TRIAL)
                                                 (CAR TRIAL]
                                  (SETQ TRIAL NEW.TRIAL]
(COND
  [(EQ (CAR TRIAL)
       (CAR NEW.VAL))
    (COND
      ((LESSP (A.GETP NEW.VAL (QUOTE PRIORITY))
              (A.GETP TRIAL (QUOTE PRIORITY)))
        (A.PUT# PLAN (LIST (QUOTE RESOURCES)
                           NAME LIMIT)
                NEW.VAL))
      (T (A.PUT# PLAN (LIST (QUOTE RESOURCES)
                            NAME LIMIT)
                 TRIAL]
  ([OR (AND FLG (GREATERP (CAR TRIAL)
                          (CAR NEW.VAL)))
       (AND (NULL FLG)
            (LESSP (CAR TRIAL)
                   (CAR NEW.VAL]
    (A.PUT# PLAN (LIST (QUOTE RESOURCES)
                       NAME LIMIT)
            TRIAL))
  (T (A.PUT# PLAN (LIST (QUOTE RESOURCES)
                        NAME LIMIT)
             NEW.VAL])
```

H. Local Duration Demons

The duration of a task can be specified by the process model or directly by the user with ENTER.DURATION. The model is also permitted to specify an estimated duration. (No provision has been made, as yet, for the user to specify an estimated duration, but there would be no difficulty in providing such a function.)

Specified durations, whatever their origins, are handled by demons. Three demons are used for this purpose. Demons using the functions SET.START and SET.END are set before any data has been entered. SET.START enters the appropriate start value whenever the end value is entered. SET.END enters an end value when a start value is given. The third demon uses the function DUR.WATCH. It is set when both a start and an end value are present, and it is fired if either is changed, making the appropriate change in the other. These functions may also be called directly by SET.DURATION if data are already present when the duration is set.

These demons do affect the data to which they are attached. Therefore they must exhibit the mutual-destruct property. The first action when either SET.START or SET.END is called must be the removal of both demons. The first action when DUR.WATCH is called must be the removal or itself. The function KILL.LOCAL.DEMON.PR is defined for this purpose. It operates on the argument list of a demon and, if that list is properly set up, removes the demon and its pair, if any. It uses KILL.LOCAL.DEMON for the actual removal. The argument list must be set up to provide the correct arguments for KILL.LOCAL.DEMON. In particular, the first term of the argument list is a list that contains the first eight arguments needed by KILL.LOCAL.DEMON. If the function is DUR.WATCH, so that there is only a single demon, the second term in the argument list is NIL. Otherwise, the second term is also a list that contains the first eight arguments of KILL.LOCAL.DEMON for the other demon. This is illustrated in Table 5, which lists the demons using SET.START and SET.END that are set on A/C-PREP at the initial construction of an empty plan. The model, in this case, specifies an estimated duration of 90 for this task.

Table 5
Demon Pair Set on A/C-PREP
on Initial Call of MAKE.PLAN

```
Demon:
     START
     NIL
     PNUMBERP
     SET.END
     Arg. list:
          (MISSION 1 TASKS PILOT-BRIEF START
                   NIL PNUMBERP SET.END)
          (MISSION 1 TASKS PILOT-BRIEF END
                   NIL PNUMBERP SET.START)
          PILOT-BRIEF
          TASKS
          DURATION
          120

Demon:
     END
     NIL
     PNUMBERP
     SET.START
     Arg. list:
          (MISSION 1 TASKS PILOT-BRIEF START
                   NIL PNUMBERP SET.END)
          (MISSION 1 TASKS PILOT-BRIEF END
                   NIL PNUMBERP SET.START)
          PILOT-BRIEF
          TASKS
          DURATION
          120
```

The final action of either SET.START or SET.END is to set a demon, using DUR.WATCH, to continue to enforce the required duration or estimated duration. The final action of DUR.WATCH is to reset the demon using itself. This demon is shown in Table 6 for PILOT-BRIEF, for which the duration is specified. Had the duration been estimated instead, the fifth element in the argument list would have been EST.DUR instead of DURATION.

Table 6
Watch Demon that Monitors Duration

```
Demon:
     T
     NIL
     DUMMY
     DUR.WATCH
     Arg. list:
          (MISSION 1 TASKS PILOT-BRIEF T NIL DUMMY
                                                   DUR.WATCH)
          NIL
          PILOT-BRIEF
          TASKS
          DURATION
          120
```

SET.START must also guard against the possibility that the start time, computed by subtracting the duration from the end time that fired the demon, may be before current time. If so, it enters the current time and then recomputes and reenters the end time. This, in turn, may initiate replanning of other tasks that may be constrained by the end time. The replanning, if needed, is handled by other demons, as discussed later.

A similar situation can occur with DUR.WATCH. If so, it is handled similarly but within CORRECT.TASK.DUR, which it calls if, after testing the existing values, it finds that the current start and end times are inconsistent with the duration.

CORRECT.TASK.DURATION must decide, first, whether to change the start or end time. For this purpose, it uses the internal priorities that have been put on the entries. As stated previously, a value that is entered via ENTER.VALUE is given priority 0, indicating that it represents an externally imposed requirement. Other values, entered through, say, SET.START or SET.END, are given priorities one higher than the priority of the value from which the new value was derived. For example, in Table 4, the start times of A/C-PREP and PILOT-BRIEF were entered and so have priorities equal to 0. Their end times, entered through SET.END, have priorities equal to 1. If these end times determined the start of FLT, and FLT had a specified duration, its end time would then have a priority of 2. The value of the priority is, thus, a measure of how far distant a given value is from the inputted requirement that indirectly caused it to be set.

CORRECT.TASK.DUR changes the start or end time of a task to obtain the desired duration, according to the priorities. Specifically, it changes the value with the highest priority. If the priorities are equal, it changes the end value.

One other convention is observed in handling priorities. If SET.START must use current time instead of the desired start time, then this start time is given priority -1. The same is true if CORRECT.TASK.DUR seeks to modify the start time of a task and cannot do so as required without violating current time. It also uses the current time with priority -1. The further convention is used that a priority of -1 will propagate as far as need be without being increased. That is, if the entry of such a value forces replanning of other tasks, the resultant values are all given priority -1. The occurrence of priority -1 anywhere in the plan is an indicator that the value is forced by the need to avoid violating current time.

Table 7 illustrates the behavior when forced by current time. Current time, in this case, is 0. We enter end times of 20 for both A/C.PREP and PILOT-BRIEF. Since the former has an estimated duration of 90, and the latter 120, this creates a conflict. The actual entries are as shown with priorities -1. Note that these values and priorities are properly carried over into the EST of FLT, and the limits set on the assignments.

Table 7
Effect when SET.START threatens a
Violation of Current Time
(Local Demons Only)

```
(enter.value 'mission 1 'tasks 'a/c-prep 'end 20]
OK
(enter.value 'mission 1 'tasks 'pilot-brief 'end 20]
OK
(print.plan 'mission 1 t]
Plan number 1 of the model named MISSION is as follows:

Tasks:
     A/C-PREP:
          Start:  0.
               Priority:  -1
               Estimated.
          End:  90.
               Priority:  -1
          EST.DUR:  90.
          UNPLANNED.SUCCESSORS:  (FLT).
          UNPLANNED.ASSIGN:  (A/C).
     PILOT-BRIEF:
          Start:  0.
               Priority:  -1
          End:  120.
               Priority:  -1
          DURATION:  120.
          UNPLANNED.SUCCESSORS:  (FLT).
          UNPLANNED.ASSIGN:  (PILOT).
     FLT:
          Start:  T.
          End:  T.
          UNPLANNED.SUCCESSORS:  (A/C-SERV PILOT-DEBRIEF).
          EST:  (120 (PRIORITY . -1)).
     A/C-SERV:
          Start:  T.
          End:  T.
          EST.DUR:  20.
          UNPLANNED.ANTECEDENTS:  (FLT).
          UNPLANNED.ASSIGN:  (A/C).
     PILOT-DEBRIEF:
          Start:  T.
          End:  T.
          DURATION:  150.
          UNPLANNED.ANTECEDENTS:  (FLT).
          UNPLANNED.ASSIGN:  (PILOT).
```

Table 7
(Continued)

```
     Unplanned tasks:
          A/C-PREP, PILOT-BRIEF, FLT, A/C-SERV,
          PILOT-DEBRIEF.
     Plannable tasks:
          FLT.
     Deferred tasks:
          A/C-PREP.
Resources:
     A/C:
          Name:  T.
          Start:  T.
          End:  T.
          UNPLANNED.CONCUR:   (A/C-SERV).
          EET:  (90 (PRIORITY . -1)).
          LST:  (0 (PRIORITY . -1) (ESTIMATED . T)).
     PILOT:
          Name:  T.
          Start:  T.
          End:  T.
          UNPLANNED.CONCUR:   (PILOT-DEBRIEF).
          EET:  (120 (PRIORITY . -1)).
          LST:  (0 (PRIORITY . -1)).
     Unplanned resources:
          A/C, PILOT.
     Plannable resources:
          A/C, PILOT.
```

Those functions that maintain the durations specified either by the model or by the user are defined as follows:

```
(SET.DURATION.DEMONS
  [LAMBDA (MODEL.NAME ID MODEL PLAN RESOURCES)
    (PROG (FLG1 FLG2 PROP LST TYPE.PLAN TYPE)
      L   [COND
            (FLG1 (SETQ PROP (QUOTE EST.DUR)))
            (T (SETQ PROP (QUOTE DURATION]
          [COND
            (FLG2 (SETQ LST (A.GETP PLAN (QUOTE RESOURCES)))
                  (SETQ TYPE.PLAN (QUOTE RES.PLAN))
                  (SETQ TYPE (QUOTE RESOURCES)))
            (T (SETQ LST (A.GETP PLAN (QUOTE TASKS)))
               (SETQ TYPE.PLAN (QUOTE TASK.PLAN))
               (SETQ TYPE (QUOTE TASKS]
          [MAPC (A.GETP MODEL (QUOTE TASKS))
                (FUNCTION (LAMBDA (X)
                    (PROG (VAL)
                          (COND
                            ((SETQ VAL (A.GETP# LST (LIST X PROP)))
                              (SET.LOCAL.DEMON PLAN TYPE X (QUOTE START)
                                               NIL
                                               (QUOTE PNUMBERP)
                                               (QUOTE SET.END)
                                               (LIST (LIST MODEL.NAME
                                                           ID TYPE X
                                                           (QUOTE START)
                                                           NIL
                                                           (QUOTE
                                                             PNUMBERP)
                                                           (QUOTE
                                                             SET.END))
                                                     (LIST MODEL.NAME
                                                           ID TYPE X
                                                           (QUOTE END)
                                                           NIL
                                                           (QUOTE
                                                             PNUMBERP)
                                                           (QUOTE
                                                             SET.START))
                                                     X TYPE PROP VAL))
```

```
                                      (SET.LOCAL.DEMON PLAN TYPE X (QUOTE END)
                                                       NIL
                                                       (QUOTE PNUMBERP)
                                                       (QUOTE SET.START)
                                                       (LIST (LIST MODEL.NAME
                                                                   ID TYPE X
                                                                   (QUOTE START)
                                                                   NIL
                                                                   (QUOTE
                                                                     PNUMBERP)
                                                                   (QUOTE
                                                                     SET.END))
                                                             (LIST MODEL.NAME
                                                                   ID TYPE X
                                                                   (QUOTE END)
                                                                   NIL
                                                                   (QUOTE
                                                                     PNUMBERP)
                                                                   (QUOTE
                                                                     SET.START))
                                                             X TYPE PROP VAL]
          (COND
            ((NULL FLG1)
              (SETQ FLG1 T)
              (GO L)))
          (COND
            ((AND (NULL FLG2)
                  RESOURCES)
              (SETQ FLG1 NIL)
              (SETQ FLG2 T)
              (GO L])

(SET.START
  [LAMBDA (OLD.VAL NEW.VAL ARG.LIST PROP LST)
    (PROG (LST PLAN PLAN.TYPE START FLG START.VAL E.FLG DEFERRED)
          [COND
            ((NULL LST)
              (SETQ LST (KILL.LOCAL.DEMON.PR ARG.LIST]
          [SETQ PLAN (A.GETP.S# PLANS (LIST (CAAR ARG.LIST)
                                            (2ND (CAR ARG.LIST]
          (COND
            ((EQUAL (2ND LST)
                    (QUOTE TASKS))
              (SETQQ PLAN.TYPE TASK.PLAN))
            (T (SETQQ PLAN.TYPE RES.PLAN)))
```

```
(COND
  ((LESSP (SETQ START (IDIFFERENCE (CAR (A.GETP NEW.VAL
                                                (QUOTE END)))
                                   (4TH LST)))
          S.CLOCK)
    (SETQ START S.CLOCK)
    (SETQ FLG T)))
[COND
  [(OR FLG (EQ (A.GETP# OLD.VAL (QUOTE (END PRIORITY)))
               -1))
    (SETQ START.VAL (LIST START (CONS (QUOTE PRIORITY)
                                      -1]
  (T (SETQ START.VAL (LIST START
                           (CONS (QUOTE PRIORITY)
                                 (ADD1 (A.GETP# NEW.VAL
                                                (QUOTE (END
                                                    PRIORITY]
(COND
  ((EQUAL (3RD LST)
          (QUOTE EST.DUR))
    (A.PUT START.VAL (QUOTE ESTIMATED)
           T)
    (SETQ E.FLG T)))
(P.ENTRY.2 (A.GETP PLAN (2ND LST))
           (CAR LST)
           (QUOTE START)
           START.VAL)
[COND
  (FLG (P.ENTRY.2 (A.GETP PLAN (2ND LST))
                  (CAR LST)
                  (QUOTE END)
                  (LIST (IPLUS S.CLOCK (4TH LST))
                        (CONS (QUOTE PRIORITY)
                              -1]
(SET.LOCAL.DEMON PLAN (2ND LST)
                 (CAR LST)
                 T NIL (QUOTE DUMMY)
                 (QUOTE DUR.WATCH)
                 (APPEND [LIST (APPEND (LDIFF (CAR ARG.LIST)
                                              (NTH (CAR
                                                      ARG.LIST)
                                                   5))
                                       (LIST T NIL (QUOTE
                                                      DUMMY)
                                             (QUOTE DUR.WATCH]
                         (LIST NIL)
                         LST))
```

```
        (COND
          (E.FLG (COND
                   ((AND [SETQ DEFERRED (A.GETP# PLAN (LIST (2ND LST)
                                                            (QUOTE
                                                             DEFERRED]
                         (MEMBER (CAR LST)
                                 DEFERRED)))
                    (T (A.ADDPROP# PLAN (LIST (2ND LST)
                                              (QUOTE DEFERRED))
                                   (CAR LST])

(SET.END
  [LAMBDA (OLD.VAL NEW.VAL ARG.LIST PROP LST)
    (PROG (LST PLAN PLAN.TYPE END.VAL E.FLG DEFERRED)
        [COND
          ((NULL LST)
            (SETQ LST (KILL.LOCAL.DEMON.PR ARG.LIST]
        [SETQ PLAN (A.GETP# PLANS (LIST (CAAR ARG.LIST)
                                        (2ND (CAR ARG.LIST]
        (COND
          ((EQUAL (2ND LST)
                  (QUOTE TASKS))
            (SETQQ PLAN.TYPE TASK.PLAN))
          (T (SETQQ PLAN.TYPE RES.PLAN)))
        [SETQ END.VAL (LIST (IPLUS (CAR (A.GETP NEW.VAL
                                                (QUOTE START)))
                                   (4TH LST))
                            (CONS (QUOTE PRIORITY)
                                  (ADD1 (A.GETP# NEW.VAL (QUOTE
                                                          (START
                                                          PRIORITY]
        (COND
          ((EQUAL (3RD LST)
                  (QUOTE EST.DUR))
            (A.PUT END.VAL (QUOTE ESTIMATED)
                   T)
            (SETQ E.FLG T)))
        (COND
          ((EQ (A.GETP# OLD.VAL (QUOTE (START PRIORITY)))
               -1)
            (A.PUT END.VAL (QUOTE PRIORITY)
                   -1)))
        (P.ENTRY.2 (A.GETP PLAN (2ND LST))
                   (CAR LST)
                   (QUOTE END)
                   END.VAL)
          (SET.LOCAL.DEMON PLAN (2ND LST)
                           (CAR LST)
                           T NIL (QUOTE DUMMY)
                           (QUOTE DUR.WATCH)
                           (APPEND [LIST (APPEND (LDIFF (CAR ARG.LIST)
                                                        (NTH (CAR
                                                               ARG.LIST)
                                                             5))
                                                 (LIST T NIL (QUOTE
                                                               DUMMY)
                                                       (QUOTE DUR.WATCH]
                                   (LIST NIL)
                                   LST))
          (COND
            (E.FLG (COND
                     ((AND [SETQ DEFERRED (A.GETP# PLAN (LIST (2ND LST)
                                                              (QUOTE
                                                               DEFERRED]
                           (MEMBER (CAR LST)
                                   DEFERRED)))
                     (T (A.ADDPROP# PLAN (LIST (2ND LST)
                                               (QUOTE DEFERRED))
                                    (CAR LST])

(DUR.WATCH
  [LAMBDA (OLD.VAL NEW.VAL ARG.LIST PROP LST)
    (PROG (PLAN T.PLAN START.VAL END.VAL DURATION NAME NEW.PROP E.FLG)
          [COND
            ((NULL LST)
              (SETQ LST (APPEND (LIST (CAAR ARG.LIST)
                                      (2ND (CAR ARG.LIST)))
                                (KILL.LOCAL.DEMON.PR ARG.LIST]
          [SETQ PLAN (A.GETP.S# PLANS (LIST (CAR LST)
                                            (2ND LST]
          (SETQ T.PLAN (A.GETP PLAN (QUOTE TASKS)))
          (SETQ START.VAL (A.GETP NEW.VAL (QUOTE START)))
          (SETQ END.VAL (A.GETP NEW.VAL (QUOTE END)))
          (SETQ DURATION (6TH LST))
          (COND
            ((EQUAL (5TH LST)
                    (QUOTE EST.DUR))
              (SETQ E.FLG T)))
```

```
(COND
  ((EQUAL (CAR END.VAL)
          (IPLUS (CAR START.VAL)
                 DURATION)))
  (T (SETQ NAME (3RD LST))
     [COND
       ((NULL (SETQ NEW.PROP (CORRECT.TASK.DUR T.PLAN NAME
                                                START.VAL
                                                END.VAL
                                                DURATION
                                                E.FLG)))
         (SETQ NEW.PROP (CORRECT.TASK.DUR T.PLAN NAME
                                            (A.GETP#
                                              T.PLAN
                                              (LIST NAME (QUOTE
                                                      TASK.PLAN)
                                                    (QUOTE
                                                      START)))
                                            (A.GETP#
                                              T.PLAN
                                              (LIST NAME (QUOTE
                                                      TASK.PLAN)
                                                   (QUOTE END)))
                                            DURATION E.FLG))
         (FIRE.LOCAL.DEMONS (A.GETP# T.PLAN (LIST NAME
                                                    (QUOTE
                                                      DEMONS)))
                            (A.GETP# T.PLAN (LIST NAME
                                                    (QUOTE
                                                    TASK.PLAN)))
                            NEW.VAL
                            (QUOTE START]
     (FIRE.LOCAL.DEMONS (A.GETP# T.PLAN (LIST NAME (QUOTE
                                                       DEMONS)))
                        (A.GETP# T.PLAN (LIST NAME
                                                (QUOTE
                                                  TASK.PLAN)))
                        NEW.VAL NEW.PROP)))
(SET.LOCAL.DEMON PLAN (4TH LST)
                 (3RD LST)
                 T NIL (QUOTE DUMMY)
                 (QUOTE DUR.WATCH)
                 ARG.LIST])
```

```
(CORRECT.TASK.DUR
  [LAMBDA (T.PLAN NAME START.VAL END.VAL DURATION E.FLG)
    (PROG (P1 P2 TRIAL VAL.LST)
          (SETQ P1 (A.GETP START.VAL (QUOTE PRIORITY)))
          (SETQ P2 (A.GETP END.VAL (QUOTE PRIORITY)))
          (COND
            [(GREATERP P1 P2)
              (SETQ TRIAL (IDIFFERENCE (CAR END.VAL)
                                       DURATION))
              (COND
                ((LESSP TRIAL S.CLOCK)
                  [SETQ VAL.LST (LIST S.CLOCK (QUOTE (PRIORITY . -1]
                  (COND
                    (E.FLG (A.PUT VAL.LST (QUOTE ESTIMATED)
                                  T)))
                  (A.PUT# T.PLAN (LIST NAME (QUOTE TASK.PLAN)
                                       (QUOTE START))
                          VAL.LST)
                  (RETURN NIL))
                (T [COND
                     [(EQ P2 -1)
                       (SETQ VAL.LST (LIST TRIAL (QUOTE (PRIORITY . -1]
                     (T (SETQ VAL.LST (LIST TRIAL (CONS (QUOTE PRIORITY)
                                                        (ADD1 P2]
                   (COND
                     (E.FLG (A.PUT VAL.LST (QUOTE ESTIMATED)
                                   T)))
                   (A.PUT# T.PLAN (LIST NAME (QUOTE TASK.PLAN)
                                        (QUOTE START))
                           VAL.LST)
                   (RETURN (QUOTE START]
            ((EQ P1 -1)
              [SETQ VAL.LST (LIST (IPLUS (CAR START.VAL)
                                         DURATION)
                                  (QUOTE (PRIORITY . -1]
              (COND
                (E.FLG (A.PUT VAL.LST (QUOTE ESTIMATED)
                              T)))
              (A.PUT# T.PLAN (LIST NAME (QUOTE TASK.PLAN)
                                   (QUOTE END))
                      VAL.LST)
              (RETURN (QUOTE END)))
          (T [SETQ VAL.LST (LIST (IPLUS (CAR START.VAL)
                                        DURATION)
                                 (CONS (QUOTE PRIORITY)
                                       (ADD1 P1]
             (COND
               (E.FLG (A.PUT VAL.LST (QUOTE ESTIMATED)
                          T)))
             (A.PUT# T.PLAN (LIST NAME
                                  (QUOTE TASK.PLAN)
                                  (QUOTE END))

                    VAL.LST)
             (RETURN (QUOTE END])

(KILL.LOCAL.DEMON.PR
  [LAMBDA (ARG.LIST)
    (PROG (NIL)
          (APPLY (QUOTE KILL.LOCAL.DEMON)
                 (APPEND (CAR ARG.LIST)
                         (LIST ARG.LIST)))
          [COND
            ((2ND ARG.LIST)
              (APPLY (QUOTE KILL.LOCAL.DEMON)
                     (APPEND (2ND ARG.LIST)
                             (LIST ARG.LIST]
          (RETURN (NTH ARG.LIST 3])

(KILL.LOCAL.DEMON
  [LAMBDA (MODEL.NAME ID TYPE NAME PROP VAL FN1 FN2 ARG.LIST)
    (A.REMVAL1# PLANS (LIST MODEL.NAME ID TYPE NAME (QUOTE DEMONS))
                (LIST PROP VAL FN1 FN2 ARG.LIST])
```

I. Type Demons

There is only one type demon currently being used. It is set by MAKE.PLAN through MAKE.PLAN.DEMONS when it calls SET.TASK.DEMONS and SET.RES.DEMONS. It uses the function FILL.PLAN to examine the limits and lists under TASKS and RESOURCES, to verify whether they are consistent and, if not, to make them so.

The form of the demon is shown in Table 8.

Table 8
Type Demon

```
(print.demonlist 'mission 1 'tasks]
General task demons:
     Demon:
          NIL
          DUMMY
          FILL.PLAN
          Arg. list:
               (MISSION 1)
               TASKS
```

Note that the form of the demon is slightly contracted, being a list of four terms, instead of five as in the local demons. The first element permits the naming of a task, resource, or other property name under TASKS or RESOURCES. It is not used here and is set to NIL. The second element is a function name and indicates the function to be used in the precondition. It is here set to DUMMY, which always returns T. Hence the demon is fired whenever it is tested. The third element is the demon function, FILL.PLAN, and the fourth its argument list. The effect is that FILL.PLAN is called whenever P.ENTRY is used.

The purpose of FILL.PLAN is to make sure that the plan, as it exists at the time, is self-consistent and has been carried as far as possible without requesting additional information from another planner, a scheduler, or the user. For this purpose, it calls a set of functions in order, and keeps recycling through this set as long as any changes are made.

The first function called by FILL.PLAN is ENTER.FIXED.DUR.TASKS. This examines, in order, each task listed in PLANNABLE. If it is also listed under DEFERRED, it is ignored. Finally, if the plan contains

AD-A047 599 SRI INTERNATIONAL MENLO PARK CALIF F/G 5/2
PLANNERS OF ACS.1.(U)
NOV 77 M C PEASE N00014-77-C-0308
UNCLASSIFIED SRI-TR-15 NL

2 OF 2
AD
A047599

END
DATE
FILMED
1-78
DDC

a value for DURATION under the task, ENTER.TASK.VALUE is called. This function is discussed later. Note that the listing of a task in PLANNABLE means that some information is available for it-- at least a limit has been set. Furthermore, if the duration is specified, no other planner is to be called; it is not further decomposed in another process model. Hence, planning of the task can proceed. It is possible that the planning of the task may have to be redone later, if the limit should be changed later by another task, but, if so, the cost of premature planning will be slight since the task is not decomposed.

The second function is ENTER.EST.DUR.TASKS, which does a similar job on the tasks for which an estimated duration has been specified. The purpose of an estimated duration is to permit bypassing the planning of a task until later. For example, it permits obtaining resource assignments before obtaining detailed plans for all tasks. This may be desirable if the coordination of assignments is expected to be the more critical requirement.

ENTER.EST.DUR.TASKS also uses ENTER.TASK.VALUE. Finally, it enters the task under DEFERRED if it is not already there.

The third function used by FILL.PLAN is CHECK.TASKS. It compares the start and end times of each task, if any, with the limits that may have been set on it. If there is a contradiction, it overrides the value with the limit, which will, in general, initiate replanning of that task. This is necessary because the limit may have been changed since the value was entered-- for example, because the start or end time of another task may have been changed. If so, that change will have changed the limit, but not the value. It is this function that maintains the self-consistency of the plan in the face of such changes.

The fourth function is CHECK.STATUS. Its purpose is to verify the PLANNABLE, UNPLANNED, and DEFERRED lists for both the tasks and resources. It does so by examining the TASK.PLAN or RES.PLAN for each task or resource and checking that the task or assignment is properly entered. This is necessary since data can be entered separately-- for example, by successive uses of ENTER.VALUE-- which completes the planning of a task or assignment in a way that would not be recognized by any of the other functions.

As stated, FILL.PLAN cycles repeatedly through these functions until there is no further change. A change initiated by adjusting the start time of a task to meet current time, for example, can therefore propagate as far as necessary through the plan.

ENTER.FIXED.DUR.TASKS and ENTER.EST.DUR.TASKS both use ENTER.TASK.VALUE. ENTER.TASK.VALUE is written somewhat more generally than is necessary for the situations that can arise here. This has been done deliberately so as not to limit future development. In particular, it considers the possibility that a given task may have both an EST and an LST, or both an EET and an LET, set on it. This could happen if an assignment were entered via ENTER.VALUE, but not if all values entered are task values. If both an EST and an LST are present, for example, it determines if one is estimated and the other not. If so, it takes the nonestimated value. If both are estimated or both are not, it takes the lower priority value, if there is one; otherwise, it takes the LST value. If only EST or LST is present but not the other, it takes that value.

A similar choice is made for the end limits, EET and LET, using the same rules except that the default condition is to choose the EET value.

If, as a result of this analysis, trial values have been set for both the start and end times, a choice is made between these by similar rules. The selected value is then entered via P.ENTRY.2. Note that the recycling feature of FILL.PLAN will eventually cause the other value to be entered also, unless there are other conditions that will change the task plan first, such as a specified duration. It is the possibility of other such conditions that leads to arranging for ENTER.TASK.VALUE to enter only a single value on a single call. Otherwise, looping could occur.

These functions are defined as follows:

```
(FILL.PLAN
  [LAMBDA (MODEL.NAME ID)
    (PROG (FLG)
          (SETQ FLG (ENTER.FIXED.DUR.TASKS MODEL.NAME ID))
          (COND
            ((ENTER.EST.DUR.TASKS MODEL.NAME ID)
              (SETQ FLG T)))
          (COND
            ((CHECK.TASKS MODEL.NAME ID)
              (SETQ FLG T)))
          (COND
            ((CHECK.STATUS MODEL.NAME ID)
              (SETQ FLG T)))
          (COND
            (FLG (FILL.PLAN MODEL.NAME ID])
(ENTER.FIXED.DUR.TASKS
  [LAMBDA (MODEL.NAME ID)
    (PROG (PLAN PLANNABLE DEFERRED FLG)
          (SETQ FLG NIL)
          (SETQ PLAN (A.GETP.S# PLANS (LIST MODEL.NAME ID)))
          [SETQ DEFERRED (A.GETP# PLAN (QUOTE (TASKS DEFERRED]
          [COND
            ([SETQ PLANNABLE (A.GETP# PLAN (QUOTE (TASKS PLANNABLE]
              (MAPC PLANNABLE (FUNCTION (LAMBDA (X)
                         (COND
                           ((AND DEFERRED (MEMBER X DEFERRED)))
                           ((A.GETP# PLAN (LIST (QUOTE TASKS)
                                                X
                                                (QUOTE DURATION)))
                             (ENTER.TASK.VALUE PLAN X)
                             (SETQ FLG T]
          (RETURN FLG])

(ENTER.EST.DUR.TASKS
  [LAMBDA (MODEL.NAME ID)
    (PROG (PLAN PLANNABLE DEFERRED FLG)
          (SETQ FLG NIL)
          (SETQ PLAN (A.GETP.S# PLANS (LIST MODEL.NAME ID)))
          [SETQ DEFERRED (A.GETP# PLAN (QUOTE (TASKS DEFERRED]
          [COND
            ([SETQ PLANNABLE (A.GETP# PLAN (QUOTE (TASKS PLANNABLE]
              (MAPC PLANNABLE (FUNCTION (LAMBDA (X)
                         (PROG (NIL)
                               (COND
                                 ([NULL (A.GETP# PLAN (LIST (QUOTE TASKS)
                                                            X
                                                            (QUOTE
                                                              EST.DUR]
                                   (RETURN NIL))
                                 ((AND DEFERRED (MEMBER X DEFERRED)))
                                 (T (SETQ FLG T)))
                               (ENTER.TASK.VALUE PLAN X]
          (RETURN FLG])
```

```
(CHECK.TASKS
  [LAMBDA (MODEL.NAME ID)
    (PROG (T.PLAN FLG TASK.LIST)
          (SETQ FLG NIL)
          [SETQ T.PLAN (A.GETP# PLANS (LIST MODEL.NAME ID (QUOTE TASKS]
          [SETQ TASK.LIST (A.GETP# MODELS (LIST MODEL.NAME (QUOTE TASKS]
          [MAPC TASK.LIST (FUNCTION (LAMBDA (X)
                     (PROG (SUBPLAN START EST LST)
                           (SETQ SUBPLAN (A.GETP T.PLAN X))
                           (COND
                             ([NULL (SETQ START (A.GETP# SUBPLAN
                                                          (QUOTE
                                                            (TASK.PLAN
                                                                     START]
                               (RETURN NIL))
                             ((EQ START T)
                               (RETURN NIL))
                             ((AND (SETQ EST (A.GETP SUBPLAN
                                                        (QUOTE EST)))
                                   (LESSP (CAR START)
                                          (CAR EST)))
                               (P.ENTRY.2 T.PLAN X (QUOTE START)
                                          EST)
                               (SETQ FLG T))
                             ((AND (SETQ LST (A.GETP SUBPLAN
                                                        (QUOTE LST)))
                                   (GREATERP (CAR START)
                                             (CAR LST)))
                               (P.ENTRY.2 T.PLAN X (QUOTE START)
                                          LST)
                               (SETQ FLG T]
          (MAPC TASK.LIST (FUNCTION (LAMBDA (X)
                     (PROG (SUBPLAN END EET LET)
                           (SETQ SUBPLAN (A.GETP T.PLAN X))
                           (COND
                             ([NULL (SETQ END (A.GETP# SUBPLAN
                                                        (QUOTE (TASK.PLAN
                                                                   END]
                               (RETURN NIL))
                             ((EQ END T)
                               (RETURN NIL))
                             ((AND (SETQ LET (A.GETP SUBPLAN
                                                        (QUOTE LET)))
                                   (GREATERP (CAR END)
                                             (CAR LET)))
                               (P.ENTRY.2 T.PLAN X (QUOTE END)
                                          LET)
                               (SETQ FLG T))
```

```
                            ((AND (SETQ EET (A.GETP SUBPLAN
                                                    (QUOTE EET)))
                                  (LESSP (CAR END)
                                         (CAR EET)))
                              (P.ENTRY.2 T.PLAN X (QUOTE END)
                                         EET)
                              (SETQ FLG T])
          (RETURN FLG])

(CHECK.STATUS
  [LAMBDA (MODEL.NAME ID)
    (PROG (PLAN MODEL RESOURCES UNPLANNED PLANNABLE DEFERRED FLG)
          (SETQ FLG NIL)
          (SETQ PLAN (A.GETP.S# PLANS (LIST MODEL.NAME ID)))
          (SETQ MODEL (A.GETP MODELS MODEL.NAME))
          [SETQ UNPLANNED (A.GETP# PLAN (QUOTE (TASKS UNPLANNED]
          [SETQ PLANNABLE (A.GETP# PLAN (QUOTE (TASKS PLANNABLE]
          [SETQ DEFERRED (A.GETP# PLAN (QUOTE (TASKS DEFERRED]
          [MAPC (A.GETP MODEL (QUOTE TASKS))
                (FUNCTION (LAMBDA (X)
                    (PROG (TASK.PLAN)
                          [SETQ TASK.PLAN (A.GETP# PLAN (LIST (QUOTE
                                                                TASKS)
                                                              X
                                                              (QUOTE
                                                              TASK.PLAN]
                          (COND
                            [(OR (EQUAL (A.GETP TASK.PLAN (QUOTE START))
                                        T)
                                 (EQUAL (A.GETP TASK.PLAN (QUOTE END))
                                        T)
                                 (A.GETP# TASK.PLAN (QUOTE (START
                                                            ESTIMATED)))
                                 (A.GETP# TASK.PLAN (QUOTE (END
                                                            ESTIMATED]
                            (T (COND
                                 ((AND UNPLANNED (MEMBER X UNPLANNED))
                                   (A.REMVAL1# PLAN (QUOTE (TASKS
                                                             UNPLANNED))
                                               X)
                                   (SETQ FLG T)))
                               (COND
                                 ((AND DEFERRED (MEMBER X DEFERRED))
                                   (A.REMVAL1# PLAN (QUOTE (TASKS
                                                             DEFERRED))
                                               X)
                                   (SETQ FLG T)))
```

```
                                    (COND
                                      ((AND PLANNABLE (MEMBER X PLANNABLE))
                                        (A.REMVAL1# PLAN (QUOTE (TASKS
                                                                  PLANNABLE))
                                                   X)
                                        (SETQ FLG T]
               [COND
                 ((SETQ RESOURCES (A.GETP MODEL (QUOTE RESOURCES)))
                   (MAPC RESOURCES (FUNCTION (LAMBDA (X)
                           (PROG (RES.PLAN)
                                 [SETQ RES.PLAN (A.GETP# PLAN (LIST (QUOTE
                                                                  RESOURCES)
                                                                    X
                                                                     (QUOTE
                                                                  RES.PLAN]
                                 (COND
                                   ((OR (EQUAL (A.GETP RES.PLAN (QUOTE
                                                                  NAME))
                                               T)
                                        (EQUAL (A.GETP RES.PLAN (QUOTE
                                                                  START))
                                               T)
                                        (EQUAL (A.GETP RES.PLAN (QUOTE
                                                                  END))
                                               T)))
                                   (T (A.REMVAL1# PLAN (QUOTE (RESOURCES
                                                                PLANNABLE))
                                                  X)
                                      (A.REMVAL1# PLAN (QUOTE (RESOURCES
                                                                UNPLANNED))
                                                  X]
               (RETURN FLG])

(ENTER.TASK.VALUE
  [LAMBDA (PLAN TASK.NAME)
    (PROG (SUBPLAN EST LST EET LET TRIAL.1 TRIAL.2)
          (SETQ SUBPLAN (A.GETP# PLAN (LIST (QUOTE TASKS)
                                            TASK.NAME)))
          (SETQ EST (A.GETP SUBPLAN (QUOTE EST)))
          (SETQ LST (A.GETP SUBPLAN (QUOTE LST)))
          (SETQ EET (A.GETP SUBPLAN (QUOTE EET)))
          (SETQ LET (A.GETP SUBPLAN (QUOTE LET)))
```

```
(COND
  [(AND EST LST)
    (COND
      ([AND (A.GETP EST (QUOTE ESTIMATED))
            (NULL (A.GETP LST (QUOTE ESTIMATED]
        (SETQ TRIAL.1 LST))
      ([AND (A.GETP LST (QUOTE ESTIMATED))
            (NULL (A.GETP EST (QUOTE ESTIMATED]
        (SETQ TRIAL.1 EST))
      ((LESSP (A.GETP EST (QUOTE PRIORITY))
              (A.GETP LST (QUOTE PRIORITY)))
        (SETQ TRIAL.1 EST))
      (T (SETQ TRIAL.1 LST]
  (EST (SETQ TRIAL.1 EST))
  (LST (SETQ TRIAL.1 LST)))
(COND
  [(AND EET LET)
    (COND
      ([AND (A.GETP EET (QUOTE ESTIMATED))
            (NULL (A.GETP LET (QUOTE ESTIMATED]
        (SETQ TRIAL.2 LET))
      ([AND (A.GETP LET (QUOTE ESTIMATED))
            (NUL (A.GETP EET (QUOTE ESTIMATED]
        (SETQ TRIAL.2 EET))
      ((LESSP (A.GETP LET (QUOTE PRIORITY))
              (A.GETP EET (QUOTE PRIORITY)))
        (SETQ TRIAL.2 LET))
      (T (SETQ TRIAL.2 EET]
  (EET (SETQ TRIAL.2 EET))
  (LET (SETQ TRIAL.2 LET)))
(COND
  [(AND TRIAL.1 TRIAL.2)
    (COND
      ([AND (A.GETP TRIAL.1 (QUOTE ESTIMATED))
            (NULL (A.GETP TRIAL.2 (QUOTE ESTIMATED]
        (P.ENTRY.2 (A.GETP PLAN (QUOTE TASKS))
                   TASK.NAME
                   (QUOTE END)
                   TRIAL.2))
      ([AND (A.GETP TRIAL.2 (QUOTE ESTIMATED))
            (NULL (A.GETP TRIAL.1 (QUOTE ESTIMATED]
        (P.ENTRY.2 (A.GETP PLAN (QUOTE TASKS))
                   TASK.NAME
                   (QUOTE START)
                   TRIAL.1))
        ((LESSP (A.GETP TRIAL.2 (QUOTE PRIORITY))
                (A.GETP TRIAL.1 (QUOTE PRIORITY)))
          (P.ENTRY.2 (A.GETP PLAN (QUOTE TASKS))
                     TASK.NAME
                     (QUOTE END)
                     TRIAL.2))
        (T (P.ENTRY.2 (A.GETP PLAN (QUOTE TASKS))
                      TASK.NAME
                      (QUOTE START)
                      TRIAL.1]
    (TRIAL.1 (P.ENTRY.2 (A.GETP PLAN (QUOTE TASKS))
                        TASK.NAME
                        (QUOTE START)
                        TRIAL.1))
    (TRIAL.2 (P.ENTRY.2 (A.GETP PLAN (QUOTE TASKS))
                        TASK.NAME
                        (QUOTE END)
                        TRIAL.2])
```

J. Assignments and Task Planning

The functions that initiate planning tasks and getting assignments are REQUEST.PLAN and REQUEST.ASSIGN. These are called by GET.PLAN, which is called, in turn, by CONT.PLAN. CONT.PLAN is the function used by the single global demon that has been defined. This demon is shown in Table 9.

Table 9
Global or Model Demon

```
(print.demonlist 'mission 1]
The model demons are:
     Demon:
          NIL
          DUMMY
          CONT.PLAN
          Arg. list:
               (MISSION 1)
```

This demon is always fired when data are added to the plan, i.e., on any use of P.ENTRY or P.ENTRY.1. It is the last demon fired by these functions, except that it may start the sequence of demon activity over. The local and type demons have done their tasks before the global demon is fired.

In the complete system, this demon can be used differently. It depends on whether or not we wish to drive the entire planning by demons. As described here, the link to other planners or schedulers is provided through REQUEST.PLAN or REQUEST.ASSIGN, which are called by GET.PLAN as described shortly. The subplan or assignment is returned as the value of this function. This means that the original planner can do nothing else until that value is returned.

An alternative procedure is to terminate CONT.PLAN on the call of GET.PLAN, rather than after using P.ENTRY.1 with the value returned by GET.PLAN. The information generated by the subplanner or scheduler, when returned via a message, would then initiate a new entry procedure similar to ENTER.VALUE, but entering the entire subplan returned. If this is done, the initiating planner relinquishes control on the execution of REQUEST.PLAN or REQUEST.ASSIGN. It is reentered when the answering message arrives. It can, in the meantime, start creating another plan, or be available for receiving additional external information about the current plan.

Whichever procedure is used, when CONT.PLAN is called, it first determines what task or assignment is to be planned next. If a function is named in the model as the value of SELECT.FN under MODEL.SPECS, that function is used. Otherwise, SELECT.NEXT is used. This is done to permit the user to define his own selection function, if necessary, to take special circumstances into account.

SELECT.NEXT returns NIL if there is no task that can be planned and no assignment that can be made. The planning process terminates in this case.

SELECT.NEXT first examines the unplanned resource assignments. If there are any that have no unplanned concurrent tasks, the first one found is chosen. It then returns (RESOURCES <resource name>). If no resource assignment can be made, it examines the unplanned tasks that have not been deferred and, if possible, returns one of these in the form (TASKS <task name>). Failing this, it examines the deferred tasks. Assignments are made as soon as possible since they may lead to conflict with other uses of the resources. However, an assignment is not sought for any resource for which the plan still shows tasks as values of UNPLANNED.CONCUR. It is not considered worth obtaining an assignment if this is true; the assignment will probably have to be changed when those tasks are planned. The exception to this rule is when it is possible to estimate a task prior to its actual planning. This, of course, is the reason for using estimated durations, and why deferred tasks are removed from the value of UNPLANNED.CONCUR for an assignment.

Failing to find an assignment that can be made, SELECT.NEXT first constructs a list of lists, each of which has the form (<task name> <number>). These sublists are created by EVALUATE. If the task has a start or end time, the number is the number of unplanned successors or antecedents, respectively. If it has no start or end time, but has either an EST or an LST, it sets N as the number of unplanned successors. If it has either an EET or an LET, it sets M as the number of unplanned antecedents. If both N and M exist, the number returned is the lesser of N and M. If one is NIL, the number returned is the other. They cannot both be NIL, or the task would not be listed as plannable.

SELECT.NEXT examines this list and chooses the sublist with the smallest number. It returns (TASKS <task name>) with the chosen task. It does this with the list of plannable tasks if it is nonempty, otherwise it uses the list of deferred tasks. In the latter case, it modifies the returned value to (TASKS <task name> (ESTIMATED . T)).

CONT.PLAN then calls GET.PLAN, using the assignment or task selected by SELECT.NEXT. GET.PLAN sorts out whether it is a resource assignment or a task plan that is to be obtained. It assembles the information relevant to the assignment or task plan, and calls REQUEST.ASSIGN or REQUEST.PLAN as appropriate, passing on the applicable information.

One item of information passed on by GET.PLAN is the name of the scheduler responsible for an assignment, or of the planner responsible for planning a task. This name is obtained from the model as the value of MODULE under the resource or task name in the TASK.SPECS or RES.SPECS.

In the full system, REQUEST.ASSIGN and REQUEST.PLAN create messages that are sent to the message handler and routed from there to the desired module or to the teletype. If no module is specified in the model, the planner sends its request to the teletype.

Here, we are limiting attention to the planners. Hence, REQUEST.ASSIGN and REQUEST.PLAN simply enter into a dialog with the user. The values they return, however, are in the standard formats for use by P.ENTRY.1:

```
        ((NAME . <name>)(START . <start time>)(END . <end time>))
```

for REQUEST.ASSIGN, and

```
        ((START . <start time>)(END . <end time>))
```

for REQUEST.PLAN.

One detail should be noted. In the dialog called by REQUEST.PLAN, the user is asked if he wants to plan the task now. If he does, the immediate action is to call KILL.DUR.DEMONS. This examines the local demons attached to the task and removes any that use SET.START, SET.END, or DUR.WATCH. REQUEST.PLAN will be called only on tasks that either have no duration specified, or for which an estimated duration is given so that the planning of the task has been deferred. In the latter case, the demons are set to maintain the estimated duration until it is planned. These demons must be removed when the task is actually planned.

Table 10 shows the effect of these functions. The process model is the same as that used before, with FLT being specified by the user as having a duration of 200. The entry of a value for the start of FLT permits the plan to be driven to completion, with the required assignments and planning of the deferred tasks being handled by consultation with the user. Table 10a shows this dialog; Table 10b shows the resultant plan.

Table 10
EFFECT OF GLOBAL DEMON

10a
DIALOG

```
(enter.value 'mission 1 'tasks 'flt 'start 500]

Need an assignment of a PILOT for MISSION number 1.
Current requirements are as follows:
     Latest start time desired:  380.
     Earliest end time desired:  850.
          Enter name of PILOT assigned,
          or NIL or ] to defer assignment:  Baker
Should the latest start and earliest end times be used?
                                                     (Y or N)  y

Need an assignment of a A/C for MISSION number 1.
Current requirements are as follows:
     Latest start time desired:  410.
     Earliest end time desired:  720.
          Enter name of A/C assigned,
          or NIL or ] to defer assignment:  12345
Should the latest start and earliest end times be used?
                                                     (Y or N)  n
Should the latest start time be used?  (Y or N)  n
          Enter start time of assignment:  400
Should the earliest end time be used?  (Y or N)  n
          Enter end time of assignment:  750

The task named A/C-PREP in plan number 1 for MISSION needs
                                                     planning.
It has been estimated as follows based on an estimated
                                                     duration of 90.
     Start time:  410.
          Priority:  1.
          Estimated.
     End time:  500.
          Priority:  0.
     Earliest start time desired:  400.
          Priority:  0.
     Latest end time desired:  500.
          Priority:  0.
Do you want to plan this task now?  (Y or ])  y
Are the start and end times OK?  (Y or N)  n
Is the start time OK?  (Y or N)  n
     Enter planned start time:  400
Is the end time OK?  (Y or N)  y
```

Table 10a
(Continued)

```
The task named A/C-SERV in plan number 1 for MISSION needs
                                                    planning.
It has been estimated as follows based on an estimated
                                                    duration of 20.
    Start time:  700.
        Priority:  0.
    End time:  720.
        Priority:  2.
        Estimated.
    Earliest start time desired:  700.
        Priority:  0.
    Latest end time desired:  750.
        Priority:  0.
Do you want to plan this task now?  (Y or ])  y
Are the start and end times OK?  (Y or N)  n
Is the start time OK?  (Y or N)  y
Is the end time OK?  (Y or N)  n
    Enter planned end time:  750
OK
```

10b
FINAL PLAN

```
(print.plan 'mission 1 t]
Plan number 1 of the model named MISSION is as follows:

Tasks:
     A/C-PREP:
          Start:  400.
               Priority:  0
          End:  500.
               Priority:  0
          EST.DUR:  90.
          LET:  (500 (PRIORITY . 0)).
          EST:  (400 (PRIORITY . 0)).
     PILOT-BRIEF:
          Start:  380.
               Priority:  1
          End:  500.
               Priority:  0
          DURATION:  120.
          LET:  (500 (PRIORITY . 2)).
          EST:  (380 (PRIORITY . 1)).
     FLT:
          Start:  500.
               Priority:  0
          End:  700.
               Priority:  1
          DURATION:  200.
          EST:  (500 (PRIORITY . 0)).
          LET:  (700 (PRIORITY . 1)).
     A/C-SERV:
          Start:  700.
               Priority:  0
          End:  750.
               Priority:  0
          EST.DUR:  20.
          EST:  (700 (PRIORITY . 0)).
          LET:  (750 (PRIORITY . 0)).
     PILOT-DEBRIEF:
          Start:  700.
               Priority:  1
          End:  850.
               Priority:  2
          DURATION:  150.
          EST:  (700 (PRIORITY . 1)).
          LET:  (850 (PRIORITY . 2)).
```

Table 10b
(Continued)

```
Resources:
     A/C:
               Name:  12345.
               Start:  400.
                    Priority:  0
               End:  750.
                    Priority:  0
               EET:  (750 (PRIORITY . 0)).
               LST:  (400 (PRIORITY . 0)).
     PILOT:
               Name:  BAKER
               Start:  380.
                    Priority:  1
               End:  850.
                    Priority:  2
               EET:  (850 (PRIORITY . 2)).
               LST:  (380 (PRIORITY . 1)).
```

The functions described in this section are as follows:

```
(CONT.PLAN
  [LAMBDA (ARG.LIST TYPE)
    (PROG (SELECT.FN NEXT PLAN SUBLAN TYPE.PLAN FLG LST1 LST2 SUBPLAN)
          [COND
            [(SETQ SELECT.FN (A.GETP# MODELS (LIST (CAAR ARG.LIST)
                                                   (QUOTE MODEL.SPECS)
                                                   (QUOTE SELECT.FN]
            (T (SETQ SELECT.FN (QUOTE SELECT.NEXT]
      L   (CHECK.STATUS (CAAR ARG.LIST)
                        (2ND (CAR ARG.LIST)))
          (COND
            ((SETQ NEXT (APPLY* SELECT.FN (SETQ PLAN (A.GETP.S# PLANS
                                                                   (CAR
                                                                 ARG.LIST)))
                                   ARG.LIST TYPE LST1 LST2)))
            (T (RETURN NIL)))
          (COND
            ((EQUAL (CAR NEXT)
                    (QUOTE TASKS))
              (SETQQ TYPE.PLAN TASK.PLAN)
              (SETQ FLG NIL))
            (T (SETQQ TYPE.PLAN RES.PLAN)
               (SETQ FLG T)))
```

```
             [SETQ SUBPLAN (GET.PLAN ARG.LIST (CAR NEXT)
                                    (2ND NEXT)
                                    (A.GETP# PLAN (LIST (CAR NEXT)
                                                        (2ND NEXT)
                                                        TYPE.PLAN
                                                        (QUOTE START)))
                                    (A.GETP# PLAN (LIST (CAR NEXT)
                                                        (2ND NEXT)
                                                        TYPE.PLAN
                                                        (QUOTE END)))
                                    (A.GETP# PLAN (LIST (CAR NEXT)
                                                        (2ND NEXT)
                                                        (QUOTE EST)))
                                    (A.GETP# PLAN (LIST (CAR NEXT)
                                                        (2ND NEXT)
                                                        (QUOTE LST)))
                                    (A.GETP# PLAN (LIST (CAR NEXT)
                                                        (2ND NEXT)
                                                        (QUOTE EET)))
                                    (A.GETP# PLAN (LIST (CAR NEXT)
                                                        (2ND NEXT)
                                                        (QUOTE LET]
             (COND
               ((AND FLG (ATOM SUBPLAN))
                 (SETQ LST2 (CONS SUBPLAN LST2))
                 (GO L))
               ((ATOM SUBPLAN)
                 (SETQ LST1 (CONS SUBPLAN LST1))
                 (GO L))
               (T (P.ENTRY.1 PLAN NEXT SUBPLAN])

(SELECT.NEXT
  [LAMBDA (PLAN ARG.LIST TYPE)
    (PROG (PLANNABLE NEXT LST FLG)
          (COND
            [[AND [SETQ PLANNABLE (A.GETP# PLAN (QUOTE (RESOURCES
                                                           PLANNABLE]
                  (SETQ NEXT (SOME PLANNABLE
                                   (FUNCTION (LAMBDA (X)
                                       (NULL (A.GETP# PLAN (LIST (QUOTE
                                                            RESOURCES)
                                                               X
                                                               (QUOTE
                                                       UNPLANNED.CONCUR]
              (RETURN (LIST (QUOTE RESOURCES)
                            (CAR NEXT]
            [(SETQ PLANNABLE (A.GETP# PLAN (QUOTE (TASKS PLANNABLE]
            ([SETQ PLANNABLE (A.GETP# PLAN (QUOTE (TASKS DEFERRED]
              (SETQ FLG T))
            (T (RETURN NIL)))
```

```
        [SETQ LST (MAPCAR PLANNABLE (FUNCTION (LAMBDA (X)
                             (EVALUATE X (A.GETP# PLAN (LIST (QUOTE
                                                               TASKS)
                                                             X]
        (SETQ NEXT (CAR LST))
        [MAPC LST (FUNCTION (LAMBDA (X)
                  (COND
                    ((LESSP (CDR X)
                            (CDR NEXT))
                      (SETQ NEXT X]
        (COND
          [FLG (RETURN (LIST (QUOTE TASKS)
                             (CAR NEXT)
                             (QUOTE (ESTIMATED . T]
          (T (RETURN (LIST (QUOTE TASKS)
                           (CAR NEXT])

(EVALUATE
  [LAMBDA (NAME SUBPLAN)
    (PROG (N M)
        [COND
          [(A.GETP# SUBPLAN (QUOTE (TASK.PLAN START)))
            (RETURN (CONS NAME (LENGTH (A.GETP SUBPLAN (QUOTE
                                                  UNPLANNED.SUCCESSORS]
          ((A.GETP# SUBPLAN (QUOTE (TASK.PLAN END)))
            (RETURN (CONS NAME (LENGTH (A.GETP SUBPLAN (QUOTE
                                                 UNPLANNED.ANTECEDENTS]
        [COND
          ((OR (A.GETP SUBPLAN (QUOTE EST))
               (A.GETP SUBPLAN (QUOTE LST)))
            (SETQ N (LENGTH (A.GETP SUBPLAN (QUOTE
                                                  UNPLANNED.SUCCESSORS]
        [COND
          ((OR (A.GETP SUBPLAN (QUOTE LET))
               (A.GETP SUBPLAN (QUOTE EET)))
            (SETQ M (LENGTH (A.GETP SUBPLAN (QUOTE
                                                 UNPLANNED.ANTECEDENTS]
        (COND
          ((NOT (NUMBERP M))
            (RETURN (CONS NAME N)))
          ((NOT (NUMBERP N))
            (RETURN (CONS NAME M)))
          ((LESSP N M)
            (RETURN (CONS NAME N)))
          (T (RETURN (CONS NAME M])
(GET.PLAN
  [LAMBDA (ARG.LIST TYPE NAME START END EST LST EET LET)
    (PROG (TYPE.SPECS MODULE FLG TARGET SUBPLAN)
          (COND
            ((EQUAL TYPE (QUOTE TASKS))
              (SETQQ TYPE.SPECS TASK.SPECS)
              (SETQQ MODULE PLANNER)
              (SETQ FLG T))
            (T (SETQQ TYPE.SPECS RES.SPECS)
               (SETQQ MODULE SCHEDULER)))
          (SETQ TARGET (A.GETP# MODELS (LIST (CAR ARG.LIST)
                                             TYPE.SPECS NAME MODULE)))
          [COND
            [FLG (SETQ SUBPLAN (REQUEST.PLAN (CAAR ARG.LIST)
                                             TARGET
                                             (2ND (CAR ARG.LIST))
                                             NAME START END EST LST EET
                                             LET
                                             (A.GETP.S#
                                               PLANS
                                               (LIST (CAAR ARG.LIST)
                                                     (2ND (CAR
                                                            ARG.LIST))
                                                     (QUOTE TASKS)
                                                     NAME
                                                     (QUOTE EST.DUR]
            (T (SETQ SUBPLAN (REQUEST.ASSIGN (CAAR ARG.LIST)
                                             TARGET
                                             (2ND (CAR ARG.LIST))
                                             NAME START END EST LST EET
                                             LET]
          (COND
            ((NULL SUBPLAN)
              (RETURN NAME))
            (T (RETURN SUBPLAN])
(REQUEST.ASSIGN
  [LAMBDA (MODEL.NAME TARGET.MODULE ID RES.NAME START END EST LST EET
                      LET)
    (PROG (NAME X Y Z LST1 P1 P2)
          [COND
            [(NULL (SETQ NAME (A.GETP# PLANS (LIST MODEL.NAME ID
                                                    (QUOTE RESOURCES)
                                                    RES.NAME
                                                    (QUOTE RES.PLAN)
                                                    (QUOTE NAME]
            ((EQ NAME T)
              (SETQ NAME NIL))
            ((LISTP NAME)
              (SETQ NAME (CAR NAME]
          (TERPRI)
          (MAPPRINQ ("Need an assignment of a " RES.NAME " for " AME
                                                MODEL.NAME " number "
                                                ID "." TERPRI
                                 "Current requirements are as follows:  "
                                                TERPRI))
          [COND
            (NAME (MAPPRINQ ((TAB 5)
                                  "Name assigned:  " NAME "." TERPRI]
          [COND
            ((LISTP START)
              (MAPPRINQ ((TAB 5)
                              "Start time:  "
                              (CAR START)
                              "." TERPRI]
          [COND
            ((LISTP END)
              (MAPPRINQ ((TAB 5)
                              "End time:  "
                              (CAR END)
                              "." TERPRI]
          [COND
            (EST (MAPPRINQ ((TAB 5)
                                  "Earliest start time desired:  "
                                  (CAR EST)
                                  "." TERPRI]
          [COND
            (LST (SETQ LST1 LST)
                 (MAPPRINQ ((TAB 5)
                                  "Latest start time desired:  "
                                  (CAR LST1)
                                  "." TERPRI]
```

```
[COND
  (EET (MAPPRINQ ((TAB 5)
                  "Earliest end time desired:  "
                  (CAR EET)
                  "." TERPRI]
[COND
  (LET (MAPPRINQ ((TAB 5)
                  "Latest end time desired:  "
                  (CAR LET)
                  "." TERPRI]
(TERPRI)
[COND
  [(NULL NAME)
    (MAPPRINQ ((TAB 10)
               "Enter name of " RES.NAME " assigned," TERPRI
               (TAB 10)
               "or NIL or ] to defer assignment:  "))
    (COND
      ((NULL (SETQ X (READ)))
        (RETURN NIL]
  (T (MAPPRINQ ("Is assignment of " NAME " OK?" TERPRI
"Enter Y if OK, N to change, or ] to exit from assignment:  "))
     (COND
       ((EQUAL (SETQ X (READ))
               (QUOTE Y))
         (SETQ X NAME))
       ((EQUAL X (QUOTE N))
         (MAPPRINQ ((TAB 10)
                    "Enter new name:  "))
         (SETQ X (READ)))
       (T (RETURN NIL]
[COND
  ((NULL X)
    (RETURN NIL))
  [(AND (LISTP START)
        (LISTP END))
    (PRIN1 "Are the start and end times OK?  (Y or N)  ")
    (COND
      ((EQUAL (READ)
              (QUOTE Y))
        (SETQ Y (CAR START))
        (SETQ Z (CAR END))
        (SETQ P1 (A.GETP START (QUOTE PRIORITY)))
        (SETQ P2 (A.GETP END (QUOTE PRIORITY)))
        (GO L]
          ((AND (NULL (LISTP START))
                (NULL (LISTP END))
                LST EET)
              (PRIN1
"Should the latest start and earliest end times be used?  (Y or N)  ")
              (COND
                ((EQUAL (READ)
                        (QUOTE Y))
                  (SETQ Y (CAR LST))
                  (SETQ Z (CAR EET))
                  (SETQ P1 (A.GETP LST (QUOTE PRIORITY)))
                  (SETQ P2 (A.GETP EET (QUOTE PRIORITY)))
                  (GO L]
        [COND
          [(LISTP START)
            (PRIN1 "Is the start time OK?  (Y or N)  ")
            (COND
              ((EQUAL (READ)
                      (QUOTE Y))
                (SETQ Y (CAR START))
                (SETQ P1 (A.GETP START (QUOTE PRIORITY)))
                (GO L1]
          (LST (PRIN1
                 "Should the latest start time be used?  (Y or N)  ")
               (COND
                 ((EQUAL (READ)
                         (QUOTE Y))
                   (SETQ Y (CAR LST))
                   (SETQ P1 (A.GETP LST (QUOTE PRIORITY)))
                   (GO L1]
        (MAPPRINQ ((TAB 10)
                   "Enter start time of assignment:  "))
        (SETQ Y (READ))
        (SETQ P1 0)
    L1  [COND
          [(LISTP END)
            (PRIN1 "Is the end time OK?  (Y or N)  ")
            (COND
              ((EQUAL (READ)
                      (QUOTE Y))
                (SETQ Z (CAR END))
                (SETQ P2 (A.GETP END (QUOTE PRIORITY)))
                (GO L]
              (EET (PRIN1
                     "Should the earliest end time be used?  (Y or N)  ")
                   (COND
                     ((EQUAL (READ)
                             (QUOTE Y))
                       (SETQ Z (CAR EET))
                       (SETQ P2 (A.GETP EET (QUOTE PRIORITY)))
                       (GO L]
            (MAPPRINQ ((TAB 10)
                       "Enter end time of assignment:  "))
            (SETQ Z (READ))
            (SETQ P2 0)
        L   (SETQ Y (LIST Y (CONS (QUOTE PRIORITY)
                                  P1)))
            (SETQ Z (LIST Z (CONS (QUOTE PRIORITY)
                                  P2)))
            (RETURN (LIST (LIST (QUOTE NAME)
                                X)
                          (CONS (QUOTE START)
                                Y)
                          (CONS (QUOTE END)
                                Z])

(REQUEST.PLAN
  [LAMBDA (MODEL.NAME TARGET.MODULE ID TASK.NAME START END EST LST EET
                     LET EST.DUR)
    (PROG (X Y LST1 P1 P2)
            (TERPRI)
            (MAPPRINQ ("The task named " TASK.NAME " in plan number " ID
                                         " for " MODEL.NAME
                                         " needs planning." TERPRI))
            [COND
              ((AND EST.DUR (LISTP START))
                (MAPPRINQ (
"It has been estimated as follows based on an estimated duration of "
                                                      EST.DUR "." TERPRI)))
              (T (MAPPRINQ ("Its current status is as follows:" TERPRI]
            [COND
              ((LISTP START)
                (MAPPRINQ ((TAB 5)
                           "Start time:  "
                           (CAR START)
                           "." TERPRI (TAB 10)
                           "Priority:  "
                           (A.GETP START (QUOTE PRIORITY))
                           "." TERPRI))
                (COND
                  ((A.GETP START (QUOTE ESTIMATED))
                    (MAPPRINQ ((TAB 10)
                               "Estimated." TERPRI]
```

```
[COND
  ((LISTP END)
    (MAPPRINQ ((TAB 5)
               "End time:  "
               (CAR END)
               "." TERPRI (TAB 10)
               "Priority:  "
               (A.GETP END (QUOTE PRIORITY))
               "." TERPRI))
    (COND
      ((A.GETP END (QUOTE ESTIMATED))
        (MAPPRINQ ((TAB 10)
                   "Estimated." TERPRI]
[COND
  (EST (MAPPRINQ ((TAB 5)
                  "Earliest start time desired:  "
                  (CAR EST)
                  "." TERPRI (TAB 10)
                  "Priority:  "
                  (A.GETP EST (QUOTE PRIORITY))
                  "." TERPRI]
[COND
  (LST (SETQ LST1 LST)
       (MAPPRINQ ((TAB 5)
                  "Latest start time desired:  "
                  (CAR LST1)
                  "." TERPRI (TAB 10)
                  "Priority:  "
                  (A.GETP LST1 (QUOTE PRIORITY))
                  "." TERPRI]
[COND
  (EET (MAPPRINQ ((TAB 5)
                  "Earliest end time desired:  "
                  (CAR EET)
                  "." TERPRI (TAB 10)
                  "Priority:  "
                  (A.GETP EET (QUOTE PRIORITY))
                  "." TERPRI]
[COND
  (LET (MAPPRINQ ((TAB 5)
                  "Latest end time desired:  "
                  (CAR LET)
                  "." TERPRI (TAB 10)
                  "Priority:  "
                  (A.GETP LET (QUOTE PRIORITY))
                  "." TERPRI]
(PRIN1 "Do you want to plan this task now?  (Y or ])  ")
(COND
  ((NULL (READ))
    (RETURN NIL)))
```

```
            (KILL.DUR.DEMONS MODEL.NAME ID TASK.NAME)
            [COND
              [(AND (LISTP START)
                    (LISTP END))
                (MAPPRINQ (TERPRI
                            "Are the start and end times OK?  (Y or N)   "))
                (COND
                  ((EQUAL (READ)
                          (QUOTE Y))
                    (SETQ X (CAR START))
                    (SETQ Y (CAR END))
                    (GO L]
              ((AND (NULL (LISTP START))
                    (NULL (LISTP END))
                    EST LET)
                (PRIN1
"Should the earliest start and latest end times be used?  (Y or N)   ")
                (COND
                  ((EQUAL (READ)
                          (QUOTE Y))
                    (SETQ X (CAR EST))
                    (SETQ Y (CAR LET))
                    (GO L]
            [COND
              [(LISTP START)
                (PRIN1 "Is the start time OK?  (Y or N)   ")
                (COND
                  ((EQUAL (READ)
                          (QUOTE Y))
                    (SETQ X (CAR START))
                    (GO L1]
              (EST (PRIN1
                     "Should the earliest start time be used?  (Y or N)   ")
                   (COND
                     ((EQUAL (READ)
                             (QUOTE Y))
                       (SETQ X (CAR EST))
                       (GO L1]
            (MAPPRINQ ((TAB 5)
                       "Enter planned start time:  "))
            (SETQ X (READ))
            (SETQ P1 0)
```

```
L1   [COND
       [(LISTP END)
         (PRIN1 "Is the end time OK?  (Y or N)   ")
         (COND
           ((EQUAL (READ)
                   (QUOTE Y))
             (SETQ Y (CAR END))
             (GO L]
       (LET (PRIN1
                "Should the latest end time be used?  (Y or N)   ")
            (COND
              ((EQUAL (READ)
                      (QUOTE Y))
                (SETQ Y (CAR LET))
                (GO L]
     (MAPPRINQ ((TAB 5)
                "Enter planned end time:  "))
     (SETQ Y (READ))
     (SETQ P2 0)
L    [COND
       ((ZEROP P1))
       [(LISTP START)
         (SETQ P1 (A.GETP START (QUOTE PRIORITY]
       (EST (SETQ P1 (A.GETP EST (QUOTE PRIORITY]
     [COND
       ((LISTP START))
       [(AND P1 LST (LESSP (A.GETP LST (QUOTE PRIORITY))
                           P1))
         (SETQ P1 (A.GETP LST (QUOTE PRIORITY]
       ((AND LST (NULL P1))
         (SETQ P1 (A.GETP LST (QUOTE PRIORITY]
     [COND
       ((ZEROP P2))
       [(LISTP END)
         (SETQ P2 (A.GETP END (QUOTE PRIORITY]
       (EET (SETQ P2 (A.GETP EET (QUOTE PRIORITY]
     [COND
       ((LISTP END))
       [(AND P2 LET (LESSP (A.GETP LET (QUOTE PRIORITY))
                           P2))
         (SETQ P2 (A.GETP LET (QUOTE PRIORITY]
       ((AND LET (NULL P2))
         (SETQ P2 (A.GETP LET (QUOTE PRIORITY]
            (COND
              ((AND P2 (NULL P1))
                (SETQ P1 (ADD1 P2)))
              ((AND P1 (NULL P2))
                (SETQ P2 (ADD1 P1)))
              ((AND (NULL P1)
                    (NULL P2))
                (SETQ P1 0)
                (SETQ P2 0)))
            (RETURN (LIST (LIST (QUOTE START)
                                X
                                (CONS (QUOTE PRIORITY)
                                      P1))
                          (LIST (QUOTE END)
                                Y
                                (CONS (QUOTE PRIORITY)
                                      P2])

(KILL.DUR.DEMONS
  [LAMBDA (MODEL.NAME ID TASK)
    (PROG (DEMONLIST)
            (COND
              ([SETQ DEMONLIST (A.GETP# PLANS (LIST MODEL.NAME ID (QUOTE
                                                                   TASKS)
                                                    TASK
                                                    (QUOTE DEMONS]
                (MAPC DEMONLIST (FUNCTION (LAMBDA (X)
                           (COND
                             ((MEMBER (4TH X)
                                    (QUOTE (DUR.WATCH SET.START SET.END)))
                               (KILL.LOCAL.DEMON.PR (5TH X])
```

K. Alert on Given Values.

One additional detail needs to be described. When a value has been entered via ENTER.VALUE, it needs to be watched. It can be changed by a conflict with current time, or as a result of other replanning. If the planner is top level, the user should be advised of the change. If it is subordinate, the planner that was the source of the value should be advised.

To handle this requirement at the top level, P.ENTRY sets a demon using the function PRINT.CHANGE.ALERT. It determines that it is at the top level by using the inverse of PRINT.SUP.FLG, assuming that PRINT.SUP.FLG will be set in any subordinate planner to suppress the final printing of "OK." A similar device can be used to indicate when another planner should be notified of a change.

The demon is shown in Table 11, where it is assumed that the plan has been given the value 200 as the start of FLT.

Table 11
Watch Demon on Inputted Value

```
Demon:
     START
     200
     CHANGE
     PRINT.CHANGE.ALERT
     Arg. list:
          (MISSION 1)
          TASKS
          FLT
          START
          200
```

The effect of this demon is illustrated in Table 12. Here, current time has been set at 150. The value entered is the start of FLT at 200, creating a conflict with current time through the specified duration of PILOT-BRIEF and the estimated duration of A/C-PREP. In Table 12, no duration has been specified for FLT, hence the planning process is truncated. Table 12a shows the alert message. Table 12b shows the resultant plan, truncated to remove the unplanned parts.

Table 12
Scenario Exhibiting a Forced
Change of Inputted Value

12a
Input Dialog

```
(enter.value 'mission 1 'tasks 'flt 'start 200]

                   **ALERT**
  *Cannot maintain inputted condition
   for MISSION number 1.*

Initial requirement was for the START of FLT to be 200.
Currently being reset to 270.

Change forced by constraint of current time.

                    ***


The task named A/C-PREP in plan number 1 for MISSION needs
                                                 planning.
It has been estimated as follows based on an estimated
                                              duration  of 90.
     Start time:  150.
          Priority:  -1.
          Estimated.
     End time:  240.
          Priority:  -1.
          Estimated.
     Latest end time desired:  270.
          Priority:  -1.
Do you want to plan this task now?  (Y or ])  ]
The task named FLT in plan number 1 for MISSION needs
                                                planning.
Its current status is as follows:
     Start time:  270.
          Priority:  -1.
     Earliest start time desired:  270.
          Priority:  -1.
Do you want to plan this task now?  (Y or ])  ]
OK
```

Table 12b
Plan

```
(print.plan 'mission 1 t]
Plan number 1 of the model named MISSION is as follows:

Tasks:
     A/C-PREP:
          Start:  150.
               Priority:  -1
               Estimated.
          End:  240.
               Priority:  -1
               Estimated.
          EST.DUR:  90.
          UNPLANNED.ASSIGN:  (A/C).
          LET:  (270 (PRIORITY . -1)).
     PILOT-BRIEF:
          Start:  150.
               Priority:  -1
          End:  270.
               Priority:  -1
          DURATION:  120.
          UNPLANNED.ASSIGN:  (PILOT).
          LET:  (270 (PRIORITY . -1)).
     FLT:
          Start:  270.
               Priority:  -1
          End:  T.
          UNPLANNED.SUCCESSORS:  (A/C-SERV PILOT-DEBRIEF).
          EST:  (270 (PRIORITY . -1) (ESTIMATED . T)).
     A/C-SERV:
          Start:  T.

               ...
```

The function that does this is as follows:

```
(PRINT.CHANGE.ALERT
  [LAMBDA (OLD.VAL NEW.VAL ARG.LIST PROP)
    (PROG (NIL)
          (COND
            ((EQUAL (A.GETP NEW.VAL PROP)
                    (A.GETP OLD.VAL PROP))
              (RETURN NIL)))
          [MAPPRINQ (TERPRI (TAB 23)
                            "**ALERT**" TERPRI (TAB 10)
                            "*Cannot maintain inputted condition "
                            TERPRI
                            (TAB 11)
                            "for "
                            (CAAR ARG.LIST)
                            " number "
                            (2ND (CAR ARG.LIST))
                            ".*"
                            (RPTQ 2 (TERPRI))
                            "Initial requirement was for the " PROP
                            " of "
                            (3RD ARG.LIST)
                            " to be "
                            (5TH ARG.LIST)
                            "." TERPRI "Currently being reset to "
                            (CAR (A.GETP NEW.VAL PROP))
                            "."
                            (RPTQ 2 (TERPRI]
          [COND
            ((EQ (A.GETP# NEW.VAL (LIST PROP (QUOTE PRIORITY)))
                 -1)
              (MAPPRINQ ("Change forced by constraint of current time."
                         (RPTQ 2 (TERPRI]
          (MAPPRINQ ((TAB 26)
                    "***"
                    (RPTQ 2 (TERPRI])
```

V. CONCLUSIONS

The planners of ACS.1 demonstrate the capability of handling a complex set of interlocking constraints, and of maintaining the consistency of the data contained in its plans according to those constraints.

The mechanism used for enforcing the continued self-consistency of the plans is a system of demons. Each of the local demons expresses a particular constraint among the separate elements of a plan. The type demons, when fired, review the plan in its entirety and make adjustments where necessary to make the data conform to the limits established by the local demons. The global demon, which is attached to the plan as a whole, drives the planning process as far as possible, given the data that have been specified for it.

The design demonstrates the feasibility of using demons to provide for the continuing enforcement of a complex set of constraints. It also demonstrates the feasibility of functions to generate a system of demons that is the active counterpart of a complex set of passively stated rules.

An additional requirement has been to provide a design that permits modifying the knowledge used by the system without major reprogramming and without requiring the user to know the details of the system's design or implementation. This is necessary to provide for system adaptation and its evolution to meet changing conditions and requirements.

To obtain the flexibility required of the system, the principle used in the planners, and throughout ACS.1, has been to express the system's knowledge as an explicit model-- in the case of the planners, as a process model. This model is encoded separately and is used to specialize the various functions that execute the required operations on the plans. These functions are designed to have a sufficient degree of generality to accommodate a variety of models. The models, therefore, are available for modification to meet new requirements or conditions, without requiring that the functions be reprogrammed.

The planners of ACS.1 demonstrate the feasibility of using a model-driven design for planning purposes, and so of permitting the user to define the structure and details of his planning requirements.
@

INDEX OF FUNCTIONS

DISTRIBUTION LIST

Defense Documentation Center Cameron Station Alexandria, Virginia 22314	12 copies
Office of Naval Research Information Systems Program Code 437 Arlington, Virginia 22217	2 copies
Office of Naval Research Code 102IP Arlington, Virginia 22217	6 copies
Office of Naval Research Branch Office, Boston 495 Summer Street Boston, Massachusetts 02210	1 copy
Office of Naval Research Branch Office, Chicago 536 South Clark Street Chicago, Illinois 60605	1 copy
Office of Naval Research Branch Office, Pasadena 1030 East Green Street Pasadena, California 91106	1 copy
New York Area Office 715 Broadway - 5th Floor New York, New York 10003	1 copy
Naval Research Laboratory Technical Information Division, Code 2627 Washington, D.C. 20375	6 copies
Dr. A. L. Slafkosky Scientific Advisor Commandant of the Marine Corps (Code RD-1) Washington, D.C. 20380	1 copy
Office of Naval Research Code 455 Arlington, Virginia 22217	1 copy
Office of Naval Research Code 458 Arlington, Virginia 22217	1 copy
Naval Ocean Systems Center Advanced Software Technology Division Code 822 San Diego, California 92152	1 copy

Mr. E. H. Gleissner Naval Ship Research & Development Center Computation and Mathematics Department Bethesda, Maryland 20084	1 copy
Captain Grace M. Hopper NAICOM/MIS Planning Branch (OP-916D) Office of Chief of Naval Operations Washington, D.C. 20350	1 copy
Mr. Kin B. Thompson Technical Director Information Systems Division (OP-91T) Office of Chief of Naval Operations Washington, D.C. 20350	1 copy
Director National Security Agency Attn: Mr. Glick Fort George G. Meade, Maryland 20755	1 copy
Naval Aviation Integrated Logistic Support Center Code 800 Patuxent River, Maryland 20670	1 copy
Professor Omar Wing Columbia University in the City of New York Department of Electrical Engineering and Computer Science New York, New York 10027	1 copy
Mr. M. Culpepper Code 183 Naval Ship Research and Development Center Bethesda, Maryland 20084	1 copy
Mr. D. Jefferson Code 188 Naval Ship Research and Development Center Bethesda, Maryland 20084	1 copy
Robert C. Kolb, Head Code 824 Tactical Command Control and Navigation Division Naval Ocean Systems Center San Diego, California 92152	1 copy

Defense Mapping Agency Topographic
  Center
Attn: Advanced Technology Division
Code 41300 (Mr. W. Mullison)
6500 Brookes Lane
Washington, D.C. 20315

Commander, Naval Sea Systems Command
Department of the Navy
Washington, D.C. 20362
Attn: (PMS30611)